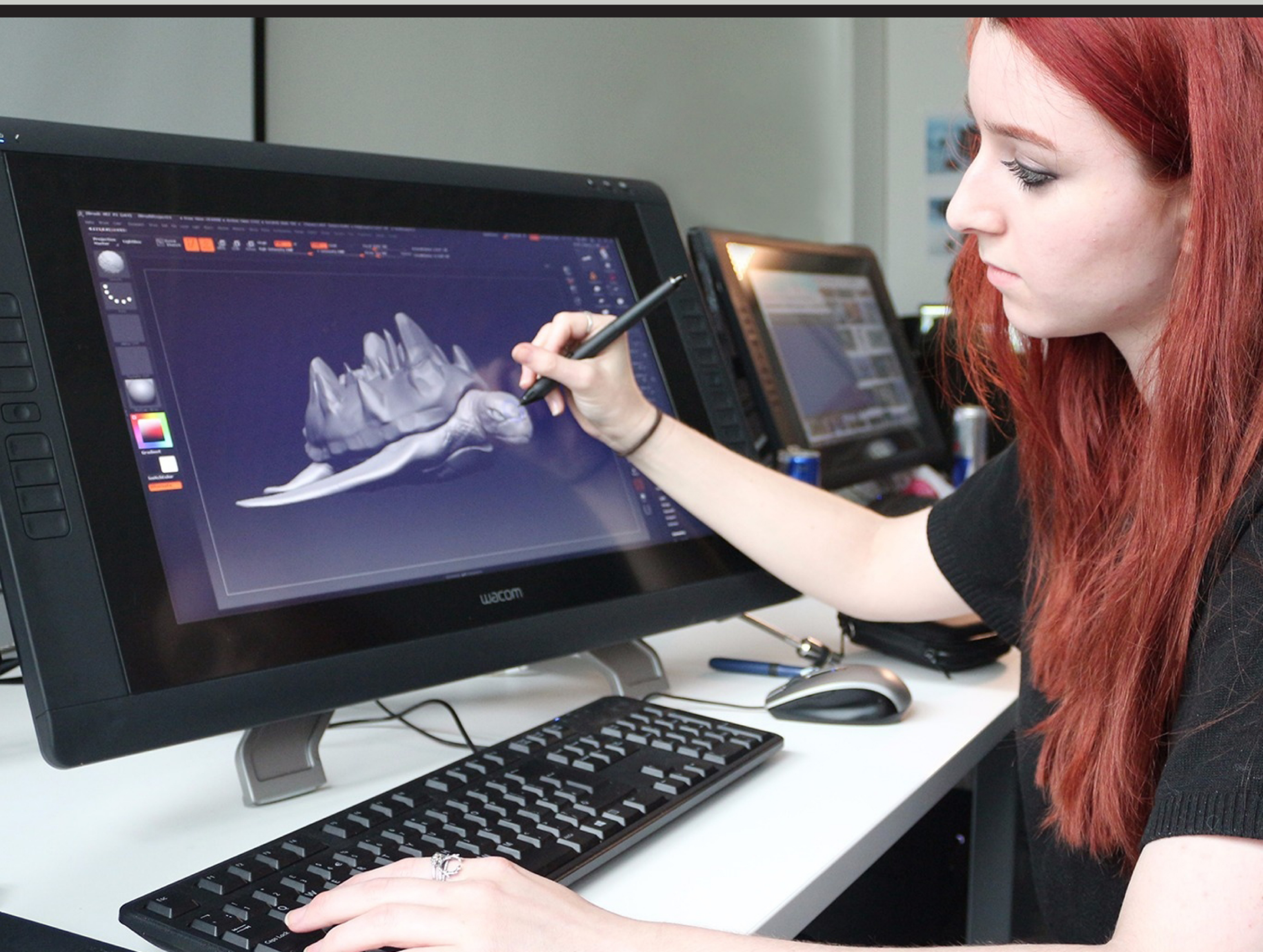
Computer Graphics and Animation

Maxwell Clements



COMPUTER GRAPHICS AND ANIMATION

COMPUTER GRAPHICS AND ANIMATION

Maxwell Clements

BIBLIOTEX
Digital Library

Computer Graphics and Animation
by Maxwell Clements

Ebook ISBN: 9781984664082

Published by:
Bibliotex
Canada
Website: www.bibliotex.com

Contents

Chapter 1	Introduction	1
Chapter 2	Sketchpad and the Design Process in Computer Graphics	20
Chapter 3	Web-based Multimedia	53
Chapter 4	Animation	85
Chapter 5	Acting and Animation	108
Chapter 6	Traditional Animation	138

1

Introduction

Interactive Evolution of Particle Systems for Computer Graphics and Animation

Content generation means creating models, levels, textures, animations, lighting, etc. for computer graphics in games, movies, and television. For media developers, content generation consumes significant time and money to produce today's complex graphics and game content. In part to address this problem, in the video game industry, it is becoming increasingly popular to provide extensive character customization tools within games and to distribute tools that allow users to create their own content outside of the game as well. Furthermore, there is a new trend towards *content generation tools as games themselves*, that is, *sandbox games* such as The Sims1, Second Life2, and Spore3. These games feature creating houses, vehicles, clothing, and creatures as primary game play features. Thus, there is a growing need for powerful and user-friendly content generation tools both to reduce the content bottleneck and further empower users.

An emerging approach to this problem is *automated content generation* through *Interactive Evolutionary Computation* (IEC), that is, automating content creation though user interaction.

This paper presents such an automated content generation method for *particle systems*, demonstrating the promise of IEC for practical content generation.

Particle systems are ubiquitous in computer graphics for producing animated effects such as fire, smoke, clouds, gunfire, water, cloth, explosions, magic, lighting, electricity, flocking, and many others. They are defined by (1) a set of points in space and (2) a set of rules guiding their behaviour and appearance, *e.g.* velocity, colour, size, shape, transparency, rotation, etc.

Since such rule sets are often complex, creating each new effect requires considerable mathematics and programming knowledge. For example, consider designing *a spherical flame shield of pulsing colours* effect for a futuristic video game or movie. Alternatively, consider designing a *particle weapon effect that fires multiple curving arcs toward the target*. In current practice, the precise mechanics for either scenario must be hand coded by a programmer. To simplify design, particle effect packages typically provide developers with a set of particle system classes, each suitable for a certain type of effect. Content developers manipulate the parameters of each particle system class by hand to produce the desired effect. The problem is that there is no way to efficiently explore the range of effects within each class.

To address this problem, this paper presents a new design, representation, and animation approach for particle systems in which (1) artificial neural networks (ANNs) control particle system behaviour, (2) the *NeuroEvolution of Augmenting Topologies* (NEAT) method produces sophisticated particle system behaviours by evolving increasingly complex ANNs, and (3) evolution is guided by user preference through an IEC interface.

Two prototype systems are discussed, NEAT Particles, a general-purpose particle effect generator, and NEAT Projectiles, which is specialized to evolve particle weapon effects for video games. Both systems interactively evolve ANNs with NEAT to control the motion and appearance of particles. An IEC interface provides a user-friendly method to evolve unique content.

In this way, NEAT Particles shows how IEC can enable practical content generation that provides an easy alternative to current, potentially cumbersome practice. In particular, NEAT Particles and NEAT Projectiles (1) enable users without programming or artistic skill to evolve unique particle system effects through a simple interface, (2) allow developers to evolve a broad range of effects within each particle class, and (3) serve as concept generators, enabling novel effect types to be easily discovered.

By allowing users to evolve particle behaviour without knowledge of physics or programming, NEAT Particles and NEAT Projectiles are a step toward the larger goal of automated content generation for games, simulations, and movies.

Background

The particle systems, IEC, and NEAT, which are components of NEAT Particles and NEAT Projectiles.

Particle Systems

The first computer-generated particle system in commercial computer graphics, called the *Genesis Effect*, appeared in Star Trek II: The Wrath of Khan. Soon after, particle systems effects became widespread on television as well. Nearly all modern video games include a particle system engine; special effects in games such as magical spells and futuristic weapons are usually implemented with particle systems.

In addition to diffuse phenomena such as fire, smoke, and explosions, particle systems can also model concrete objects such as dense trees in a forest, folded cloth and fabric, and simulated fluid motion. Realistic particle movement is often achieved by simulating real-world physics. At a more abstract level, particle systems can simulate animal and insect flocking as well as swarming behaviour. The prevalence and diversity of particle system applications demonstrates their importance to computer graphics in modern media and games.

Interactive Evolutionary Computation (IEC)

IEC is an approach to evolutionary computation (EC) in which human evaluation replaces the fitness function. A typical IEC application presents to the user the current generation of content. The user then interactively determines which members of the population will reproduce and the IEC application automatically generates the next generation of content based on the user's input. Through repeated rounds of content generation and fitness assignment, IEC enables unique content to evolve that suits the user's preferences. In some cases such content cannot be discovered or created in any other way.

IEC aids especially in evolving content for which fitness functions would be difficult or impossible to formalize (*e.g.* for aesthetic appeal). Thus, graphical content generation is a common application of IEC. IEC was first introduced in Biomorphs, which aims to illustrate theories about natural evolution. Biomorphs are patterns encoded as *Lindenmayer Systems* (Lsystems), *i.e.* grammars that specify the order in which a set of replacement rules are carried out.

Representations in *genetic art* (*i.e.* IEC applied to art) often vary, including linear or non-linear functions, fractals, and automata. Some notable examples include (1) Mutator, a cartoon and facial animation system, (2) SBART, a two-dimensional art exploration tool, (3) a tool that evolves implicit surface models such as fruits and pots, and (4) a system for evolving quadric models used as machine components. A progression of four user-selected parents in the evolution of a spaceship with a genetic art tool. In the example, the user starts by selecting a simple image that vaguely resembles what they wish to create and continues to evolve more complex images through selection until satisfied with the result. The sequence of images demonstrates the potential of IEC as an engine for content generation. These images, from Delphi NEAT Genetic Art (DNGA), are produced by ANNs evolved by NEAT.

NeuroEvolution of Augmenting Topologies

The NEAT method was originally developed to solve control and sequential decision tasks. The ANNs evolved with NEAT can control agents that select actions based on their sensory inputs. While previous methods that evolved ANNs (*i.e.* neuroevolution methods) evolved either fixed topology networks, or arbitrary random-topology networks, NEAT is the first to begin evolution with a population of small, simple networks and *complexify* the network topology into diverse species over generations, leading to increasingly sophisticated behaviour. Compared to traditional reinforcement learning techniques, which predict the long-term reward for taking actions in different states, the recurrent networks that evolve in NEAT are robust in continuous domains and in domains that require memory, making many applications possible. In this paper, particle systems are controlled by ANNs evolved by NEAT. NEAT is well-suited to this task because (1) it is a proven method for evolving ANNs, and (2) it was employed successfully in prior genetic art applications.

NEAT is based on three key principles. First, in order to allow ANN structures to increase in complexity over generations, a method is needed to keep track of which gene is which. Otherwise, it is not clear in later generations which individual is compatible with which, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique *historical marking* to every new piece of network structure that appears through a structural mutation.

The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover without the need for expensive topological analysis.

That way, genomes of different organizations and sizes stay compatible throughout evolution, solving the previously open problem of matching different topologies in an evolving population.

Second, traditionally NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before competing with other niches in the population. NEAT uses the historical markings on genes to determine to which species different individuals belong. However, in this work, because a human performs selection rather than an automated process, the usual speciation procedure in NEAT is unecessary.

Third, unlike other systems that evolve network topologies and weights, NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions and finds the appropriate complexity level for the problem.

This process of complexification has important implications for search. While it may not be practical to find a solution in a high-dimensional space by searching in that space directly, it may be possible to find it by first searching in lower dimensional spaces and complexifying the best solutions into the high-dimensional space. For IEC, complexification means that content can become more elaborate and intricate over generations.

Since its inception, NEAT has been applied to a broad array of research areas. Most notable for the approach in this paper is NERO, an interactive, realtime war game in which ANN-controlled soldiers are evolved. Because NEAT is a strong method for evolving controllers for dynamic physical systems, it can naturally be extended to evolve the motion of particles in particle effects as well.

Approach - Neat Particles

NEAT Particles combines IEC and NEAT to enable users to evolve complex particle systems. ANNs control particle system behaviour, NEAT evolves the ANNs, and an IEC interface gives the user control over evolution. NEAT Particles consists of five major components: 1) particle systems, 2) ANNs, 3) physics, 4) rendering, and 5) evolution.

Particle System Representation

A particle system is specified by an absolute *system position* in three-dimensional space and a set of particles. Each individual particle is defined by its position, velocity, colour, and size. Particle lifespan unfolds in three phases.

- At birth particles are introduced into space relative to system position and according to a *generation shape* that defines the volume within which new particles may spawn.
- During its lifetime, each particle changes and moves according to a set of rules, *i.e.* an *update function.*

- Each particle dies, and is removed from the system, when its *time to live* has expired.

NEAT Particles effects are divided into *classes* for two primary reasons: (1) user convenience and (2) performance. First, to evolve effects in a reasonable time frame, it is helpful to divide the search space for the user. Second, effects may be highly dependent upon certain variables, and unaffected by other variables. For performance reasons, it is not feasible to evolve all possible particle variables simultaneously. A better approach is implemented in NEAT Particles, in which only key variables are evolved in each particle effect class. Five particle system classes are implemented in NEAT Particles to facilitate evolving a variety of common types of effects.

- The *generic system* models effects such as fire, smoke, and explosions. Each particle has a position, velocity, colour, and size.
- The *plane system* warps individual particles into different shapes for bright flashes, lens flares, and engine exhaust effects. A single particle in the plane system is represented by four points, each of which has position, velocity, and colour.
- The *beam system* models beam, laser, or electricity effects using Bezier curves. Each particle in the beam system is a control point for the Bezier curve, including its position, velocity, and colour attributes.
- The *rotator system* models effects whose primary behaviour is orbital rotation, common in many applications. Each particle in a rotator system has rotation, position, and colour attributes.

- The *trail system* behaves similarly to the generic system, but additionally drops a trail of static particles behind each moving particle.

By providing an array of particle system classes, NEAT Particles allows designers to evolve a substantial variety of effects while conveniently constraining the search space during any particular run.

Artificial Neural Network Implementation

ANNs control particle behaviour in NEAT Particles for two primary reasons. First, ANNs are a proven method for autonomous control. Second, NEAT is a powerful method for evolving ANNs for control and sequential decision tasks. An important question is why evolving ANNs is preferable to directly evolving the variables of a traditional particle system implementation. While feasible, such an approach still ultimately relies on hand-coded rules (which constitute such systems), which thus depend on programmers to make the search possible. For example, in a traditional particle system implementation, when a new effect class is needed it requires programmers to define the effect parameters (*e.g.* colour change, motion pattern physics, etc.). In contrast, in NEAT Particles the effects of any class are represented by the same structure: ANNs.

The ANN for each particle effect dictates the characteristics and behaviour of the system. Therefore, each particle effect class includes its own ANN input and output configuration. In NEAT Particles, the ANN replaces the math and physics rules that must be programmed in traditional particle systems. Because special effects in most movie and game graphics need to be visually appealing yet not necessarily physically plausible, ANNs do not need to equate to physically realistic models. However, evolved ANN-controlled particle behaviours (*e.g. spin in a spiral while changing colour from green to orange*) are still compatible with rules in physically accurate particle simulations such as gravity, friction, or collision. Every particle in a system is guided by the same ANN. However, the ANN is activated separately for each particle. During every frame of animation in NEAT Particles an update function is executed that (1) loads inputs, (2) activates the ANN, and (3) reads outputs. The ANN outputs determine particle behaviour for the current frame of animation.

An appropriate set of inputs and outputs is associated with each effect class as follows.

The primary inputs in NEAT Particles are position and distance from centre of the system. The main outputs are velocity and colour. These are good inputs and outputs because they can encode significant variety over the long term. However, because animation happens in real-time, the change in position and distance from centre are small from one frame to the next, producing incremental changes that look smooth.

The generic particle system ANN takes the current position of the particle (px, py, pz) and distance from the centre of the system (dc) as inputs. Distance from centre introduces the potential for symmetry by allowing particles to move in relation to the system centre. The outputs are the velocity (vx, vy, vz) and colour (R, G, B) of the particle for the next frame of animation. The generic particle system produces behaviours suitable for explosions, fire, and smoke effects.

Each particle in the plane system consists four co-planar points that may be warped into different shapes. Because the corners must be coplanar for rendering purposes, the y component of velocity for each corner is fixed. Thus, the inputs to the plane system ANN are the position of each corner (px, pz) and the distance from the centre of the plane (dc). The warped quads of plane systems are commonly found in explosions, engine thrust, and glow effects.The beam system ANN controls directed beam effects. To produce twisting beams, a Bezier curve is implemented with mobile control points directed by the ANN. The inputs are the position of each Bezier control point (px, py, pz) and distance of the control point from a the centre of the system (dc). The outputs are the velocity (vx, vy, vz) and colour (R, G, B) of the control point for the next frame of animation. Beam systems produce curving, multi-coloured beams typically found in futuristic weapons, magic spells, lightning, and energy effects.

The rotator system enables evolving rotationbased effects. The inputs to the ANN are particle position (px, py, pz) and distance from the centre of the system (dc). The outputs are rotation around the x, y, and z axes (rx, ry, rz) and colour (R, G, B). Rotation-based particle systems are common in explosions, halos, and energy effects.

The trail system behaves similarly to the generic system yet provides a more complex visual effect by periodically dropping stationary particles that shrink and fade out. Therefore, the trail system ANN takes the same inputs and emits the same outputs as the generic ANN. Trail systems are convenient because they provide a computationally inexpensive form of motion blur or visual trail behind moving objects. ANNs control particle behaviour and ANN input/ outputs divide effects into classes, which shrinks the search space for users. While ANN topology and weights significantly contribute to particle behaviour, activation functions within each node play an important role as well.

Activation Functions

Unlike traditional ANNs, NEAT Particles ANN hidden nodes and output nodes contain an activation function selected from a set of eight possibilities. Theoretically, ANNs with a single activation function can evolve any behaviour ; however, multiple activation functions are preferable in NEAT Particles because the user can obtain variety more quickly and thereby evolve toward the intended effect sooner.

Physics

Each frame of animation, after the ANN is activated, the velocity for each particle is determined by the outputs. To animate a particle each frame (*i.e.* move the particle through space) a linear motion model calculates the position of the particle at time t based on *time elapsed* _ since the last frame of animation:

Pt = Pt – 1 + V _ s,

where Pt is the particle's new position vector, Pt–1 is the particle's position vector in the previous animation frame, V is the particle's velocity vector, and s is a scaling value to adjust the speed of animation.

Rendering

NEAT Particles renders particles to the screen with *billboarding,* a technique in which two-dimensional bitmap textures are mapped onto a plane (*i.e.* a *quad*) that faces perpendicular to the camera. The corners of the quad are offsets from the particle position. By facing the quad toward the camera the billboarding method convincingly conveys the illusion of translucent three-dimensional particles in space. The billboarding technique is implemented in NEAT Particles because it is the most common and versatile method to render particles. An alternative particle rendering method is point sprites ; however, they do not allow arbitrary warping of particle shape required for the beam and plane systems. There are several ways to optimize particle system rendering including level of detail (LOD), batch rendering, and GPU acceleration. NEAT Particles is compatible with all such methods; however they are not explored in this implementation.

Evolution

Evolution in NEAT Particles follows a similar procedure to other IEC applications. The user is initially presented a population of nine randomized particle systems represented by simple ANNs. Each individual system and its ANN can be inspected by *zooming in* on the system. If the initial population of nine systems is unsatisfactory, a new random batch of effects can be generated by restarting evolution. The user begins evolution by selecting a single system from the population to spawn a new generation. A population of eight new systems (*i.e.* offspring) is then generated from the ANN of the selected system (*i.e.* parent) by mutating its connection weights and possibly adding new nodes and connections. That is, offspring complexify following the NEAT method. Evolution proceeds with repeated rounds of selection and offspring production until the user is satisfied with the results. If the user is unsatisfied with an entire new generation, an *undo* function recalls the previous generation. Specifically, each new generation preserves the parent exactly and the other eight members of the population are mutated from the parent.

For each offspring, a uniformly random number of connections (between one and the number of connections in the network) are mutated by a uniformly random value between “0.5 and 0.5. Adding new nodes and connections is controlled by separate mutation rates. The probability of adding a new connection is 0.3 and the probability of adding a new node is 0.2. New nodes are assigned a random activation function and connected into the existing ANN. These parameters were found to be effective for IEC in preliminary experimentation.

Through complexification, particle system effects become increasingly sophisticated as evolution progresses. Thus, complex and unique effects are discovered that follow user preferences. The explains evolving particle system content for a more specialized purpose, weapons effects for video games.

Neat Projectiles

NEAT Projectiles is an extension of NEAT Particles designed to evolve particle weapon effects for video games. The aim is to exhibit a concrete, practical application of NEAT Particles that can potentially enhance content generation in existing real-world products. NEAT Projectiles uses similar rendering, physics, and activation functions as NEAT Particles. Furthermore, the same IEC interface drives evolution. The major differences are (1) the projectile classes, (2) the projectile constraints, and (3) the ANN inputs and outputs.

Projectile Classes

Three classes of weapon-like systems are implemented in NEAT Projectiles to mirror common weapon models in video games: (1) *dumb weapons*, (2) *directed weapons*, and (3) *smart weapons*. Dumb weapons fire simple, non-target aware projectiles and exhibit a fixed behaviour in flight. Directed weapons fire projectiles that may be steered by the user during flight. Smart weapons see the target; like a heat-seeking missile, the in-flight behaviour of smart projectiles is influenced by target motion.

Projectile Constraint

Particle weapons provide two new significant constraints on particle motion beyond generic particle effects. First, to avoid weapons firing backward, projectile velocity is limited to overall forward motion. Second, evolved projectile weapons fire in the same pattern regardless of what direction the weapon is facing. It would not make sense for projectiles emitted from a weapon to behave differently when a user points the weapon in different directions. Therefore, projectile coordinates are defined relative to the heading of the gun when it is fired. The new projectile classes and constraint mechanisms also influence the interpretation of NEAT Projectiles ANNs, as explained next.

Projectile ANNs

Because there is more than one way to make particles act as projectiles, two approaches are implemented and tested in NEAT Particles: (1) the *offset-constrained model* and (2) the *force constrained model*. In the offset-constrained model, a 90° *offset cone* in front of each particle is computed in each frame. The outputs from each particle's ANN represent a vector within the offset cone, which becomes the particle's new velocity. Offset angles are computed differently for each weapon type. A particle fired from the *dumb weapon* has a fixed offset in the direction the gun was facing on discharge. The *directed weapon* allows the user to influence projectiles while in flight; therefore particle offset is constrained to a 90° cone around the vector the weapon is currently facing. Particles fired from the *smart weapon* seek their target. Therefore, the smart particle's offset is constrained to the 90° cone around a vector from the projectile to the target.

In the force-constrained model, the ANN is similar to that used in the generic system of NEAT Particles; however a push force is applied to constrain particle movement to a general direction. The direction of the push force depends on the weapon type. The dumb weapon projectile is pushed in the direction of the gun when it discharges. The directed projectile pushes in the direction the gun is currently facing. The smart weapon pushes projectiles in the direction of the target. The combination of constraint model, classes, and correct ANN design minimizes defective offspring while allowing a sufficiently large variety of unique weapons to evolve, which is integral to efficiently producing useful content though IEC.

Experimental Results

NEAT Particles and NEAT Projectiles work in practice to produce useful particle system content. All particle systems reported were evolved in between five and ten minutes and between 20 and 30 user-guided generations. The starting point is a single curving beam to the target, which is marked with a cross. During evolution the beam splits. Finally, the desired effect is achieved with two stylized, parallel arcs that track the target. Preliminary testing of both NEAT Projectiles constraint models suggests that, compared to the force-constrained model, the offset-constrained model over-constrains evolution. It generates less variety in evolved weapon effects. However, unlike the force-constrained model, it also produces no offspring that fire back at the user. Thus, both models have their pros and cons.

Comparisons

To compare the quality of IEC particle effects to those generated by traditional methods, two hand-coded particle emitters were implemented with the same rendering method as NEAT Particles. The resulting effects exhibit similar visual quality; however, they are limited to simple behaviours because the behavioral complexity of hand-coded particle systems is dependent upon mathematics, physics, and programming, which become increasingly difficult to coordinate through hand-coded policies as more is added. Another interesting comparison can be drawn with the IEC fireworks application by Tsuneto, which produces a specialized class of particle effects. In this system, fireworks are defined by real-world attributes such as powder type, explosive payload, number of stages, stage configuration, etc. A rule-based physics system defines the behaviour of fireworks based on these attributes. Through repeated selection in an IEC interface, users can evolve fireworks to suit their preferences. Thus, unlike NEAT Particles, this system demonstrates evolving the variables of a rule system. In contrast, NEAT Particles evolves the behaviour rules themselves. Both approaches offer unique advantages. The special rule set of the fireworks application allows it to focus on a specific class of effects. NEAT Particles in contrast can evolve effects in a large variety of classes because of its generality and lack of domain-specific parameters.

2

Sketchpad and the Design Process in Computer Graphics

Construction of a drawing with Sketchpad is *itself* a model of the design process. The locations of the points and lines of the drawing model the variables of a design, and the geometric constraints applied to the points and lines of the drawing model the design constraints which limit the values of design variables. The ability of Sketchpad to satisfy the geometric constraints applied to the parts of a drawing models the ability of a good designer to satisfy all the design conditions imposed by the limitations of his materials, cost, *etc.* In fact, since, designers in many fields produce nothing themselves but a drawing of a part, design conditions may well be thought of as applying to the drawing of a part rather than to the part itself. When such design conditions are added to Sketchpad's vocabulary of constraints, the computer will be able to assist a user not only in arriving at a nice looking drawing, but also in arriving at a sound design.

Present Usefulness

As more and more applications have been made, it has become clear that the properties of Sketchpad drawings make them most useful in four broad areas:

For storing and updating drawings: Each time a drawing is made, a description of that drawing is stored in the computer in a form that is readily transferred to magnetic tape. A library of drawings will thus develop, parts of which may be used in other drawings at only a fraction of the investment of time that was put into the original drawing.

For gaining scientific or engineering understanding of operations that can be described graphically: A drawing in the Sketchpad system may contain explicit statements about the relations between its parts so that as one part is changed the implications of this change become evident throughout the drawing. For instance, Sketchpad makes it easy to study mechanical linkages, observing the path of some parts when others are moved.

As a topological input device for circuit simulators, etc.: Since, the storage structure of Sketchpad reflects the topology of any circuit or diagram, it can serve as an input for many network or circuit simulating Programmes. The additional effort required to draw a circuit completely from scratch with the Sketchpad system may well be recompensed if the properties of the circuit are obtainable through simulation of the circuit drawn.

For highly repetitive drawings: The ability of the computer to reproduce any drawn symbol anywhere at the press of a button, and to recursively include subpictures within subpictures makes it easy to produce drawings which are composed of huge numbers of parts all similar in shape.

Ring Structure

The basic *n*-component element structure described by Ross has been somewhat expanded in the implementation of Sketchpad so that all references made to a particular *n*-component element or block are collected together by a string of pointers which originates within that block. For example, not only may the end points of a line segment be found by following pointers in the line block (*n*-component element), but also all the line segments which terminate on a particular point may be found by following a string of pointers which starts within the point block. This string of pointers closes on itself; the last pointer points back to the first, hence the name "ring." The ring points both ways to make it easy to find both the next and the previous member of the ring in case, as when deleting, some change must be made to them.

Basic Operations

The basic ring structure operations are:

- Inserting a new member into a ring at some specified location on it, usually first or last.
- Removing a member from a ring.
- Putting all the members of one ring, in order, into another at some specified location in it, usually first or last.
- Performing some auxiliary operation on each member of a ring in either forward or reverse order.

These basic ring structure operations are implemented by short sections of Programme defined as MACRO instructions in the compiler language. By suitable treatment of zero and one member rings, the basic Programmes operate without making special cases.

Subroutines are used for setting up new *n*-component elements in free spaces in the storage structure. As parts of the drawing are deleted, the registers which were used to represent them become free. New components are set up at the end of the storage area, lengthening it, while free blocks are allowed to accumulate. Garbage collection periodically compacts the storage structure by removal of the free blocks.

Generic Structure, Hierarchies

The main part of Sketchpad can perform basic operations on any drawing part, calling for help from routines specific to particular types of parts when that is necessary. For example, the main Programme can show any part on the display system by calling the appropriate display subroutine. The big power of the clear-cut separation of the general and the specific is that it is easy to change the details of specific parts of the Programme to get quite different results without any need to change the general parts.

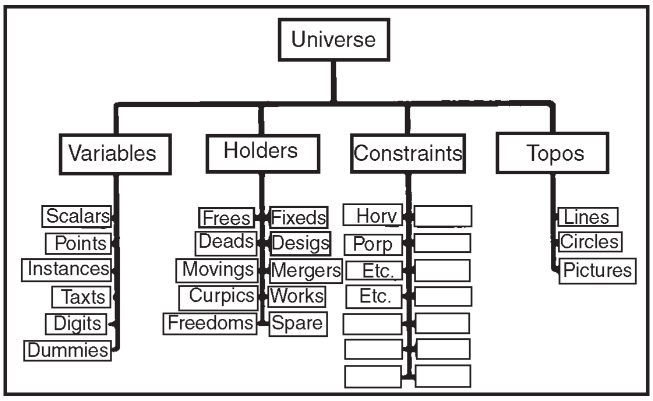


Figure : Generic Structure. The *n*-component Elements for each Point or line, *etc.* are Collected under the Generic Blocks "Lines," "Points," *etc.* Shown.

In the data storage structure the separation of general and specific is accomplished by collecting all things of one type together in a ring under a generic heading. The generic heading contains all the information which makes this type of thing different from all other types of things. Thus the data storage structure itself contains all the specific information. The generic blocks are further gathered together under super-generic or generic-generic blocks.

Expanding Sketchpad

Addition of new types of things to the Sketchpad system's vocabulary of picture parts requires only the construction of a new generic block (about 20 registers) and the writing of appropriate subroutines for the new type. The subroutines might be easy to write, as they usually are for new constraints, or difficult to write, as for adding ellipse capability, but at least a finite, well-defined task faces one to add a new ability to the system. Without a generic structure it would be almost impossible to add the instructions required to handle a new type of element.

Light Pen

In Sketchpad the light pen is time shared between the functions of coordinate input for positioning picture parts on the drawing and demonstrative input for pointing to existing picture parts to make changes. Although almost any kind of coordinate input device could be used instead of the light pen for positioning, the demonstrative input uses the light pen optics as a sort of analog computer to remove from consideration all but a very few picture parts which happen to fall within its field of view, saving considerable Programme time. Drawing systems using storage display devices of the Memotron type may not be practical because of the loss of this analog computation feature.

Pen tracking

To initially establish pen tracking, the Sketchpad user must inform the computer of an initial pen location. This has come to be known as "inking-up" and is done by "touching" any existing line or spot on the display, whereupon the tracking cross appears. If no picture has yet been drawn, the letters INK are always displayed for this purpose. Sketchpad uses loss of tracking as a "termination signal" to stop drawing. The user signals that he is finished drawing by flicking the pen too fast for the tracking Programme to follow.

Demonstrative use of Pen

During the 90% of the time that the light pen and display system are free from the tracking chore, spots are very rapidly displayed to exhibit the drawing being built, and thus the lines and circles of the drawing appear. The light pen is sensitive to these spots and reports any which fall within its field of view.

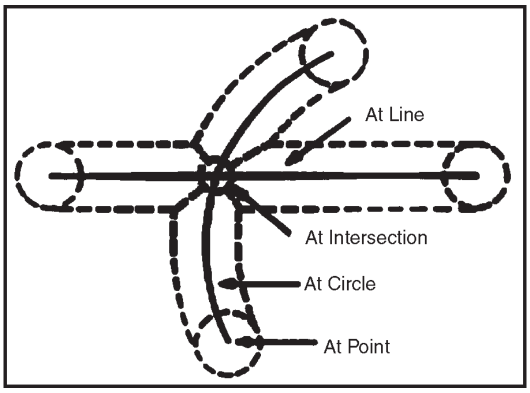


Figure : Areas in Which Pen must lie to "aim at" Existing Drawing Parts (solid lines).

The one-half inch diameter field of view of the light pen, although well suited to tracking, is relatively large for pointing. Therefore, the Sketchpad system will reject any seen part which is further from the centre of the light pen than some small minimum distance; about 1/8 inch was found to be suitable. For every kind of picture part some method must be provided for computing its distance from the light pen centre or indicating that this computation cannot be made.

After eliminating all parts seen by the pen which lie outside the smaller effective field of view, the Sketchpad system considers objects topologically related to the ones actually seen. End points of lines and attachment points of instances (subpictures) are especially important. One can thus aim at the end point of a line even though only the line is displayed. The various regions within which the pen must lie to be considered aimed at a line segment, a circle arc, their end points, or their intersection.

Pseudo Pen Location

When the light pen is aimed at a picture part, the exact location of the light pen is ignored in favour of a "pseudo pen location" exactly on the part aimed at. If no object is aimed at, the pseudo pen location is taken to be the actual pen location.

The pseudo pen location is displayed as a bright dot which is used as the "point of the pencil" in all drawing operations. As the light pen is moved into the areas outline the dot will lock onto the existing parts of the drawing, and any moving picture parts will jump to their new locations as the pseudo pen location moves to lie on the appropriate picture part. With just the basic drawing creation and manipulation functions of "draw," "move," and "delete," and the power of the pseudo pen location and demonstrative language

Programmes, it is possible to make fairly extensive drawings. Most of the constructions normally provided by straight edge and compass are available in highly accurate form. Most important, however, the pseudo pen location and demonstrative language give the means for entering the topological properties of a drawing into the machine.

Display Generation

The display system, or “scope,” on the TX-2 is a ten bit per axis electrostatic deflection system able to display spots at a maximum rate of about 100,000 per second. The coordinates of the spots which are to be seen on the display are stored in a large table so that computation and display may proceed independently. If, instead of displaying each spot successively, the display Programme displays them in a random order or with interlace, the flicker of the display is reduced greatly.

Marking of Display File

Of the 36 bits available to store each display spot in the display file, 20 give the coordinates of that spot for the display system, and the remaining 16 give the address of the *n*-component element which is responsible for adding that spot to the display. Thus, all the spots in a line are tagged with the ring structure address of that line, and all the spots in an instance (sub-picture) are tagged as belonging to that instance. The tags are used to identify the particular part of the drawing being aimed at by the light pen. If a part of the drawing is being moved by the light pen, its display spots will be recomputed as quickly as possible to show it in successive positions. The display spots for such moving parts are stored at the end of the display file so that the display of the many non--moving parts need not be disturbed. Moving parts are made invisible to the light pen.

Magnification of Pictures

The shaft position encoder knobs are used to tell the Programme to change the display scale factor or the portion of the page displayed. The range of magnification of 2000 available makes it possible to work, in effect, on a 7-inch square portion of a drawing about ¼ mile on a side. For a magnified picture, Sketchpad computes which portion(s) of a curve will appear on the display and generates display spots for those portions only. The "edge detection" problem is the problem of finding suitable end points for the portion of a curve which appears on the display. In concept the edge detection problem is trivial. In terms of Programme time for lines and circles the problem is a small fraction of the total computational load of the system, but in terms of Programme logical complexity the edge detection problem is a difficult one. For example, the computation of the intersection of a circle with any of the edges of the scope is easy, but computation of the intersection of a circle with all four edges may result in as many as eight intersections, some pairs of which may be identical, the scope corners. Now which of these intersections are actually to be used as starts of circle arcs?

Line and Circle Generation

All of Sketchpad's displays are generated from straight line segments, circle arcs, and single points.

The generation of the lines and circles is accomplished by means of the difference equations:

$$x_i = x_{i-1} + \Delta x \qquad y_i = y_{i-1}\Delta y \qquad (1)$$

for lines, and

$$x_i = x_{i-2} + \frac{2}{R}(y_{i-1} - y_c)$$
$$y_i = y_{i-2} - \frac{2}{R}(x_{i-1} - x_c) \qquad (2)$$

for circles, where subscripts *i* indicate successive display spots, subscript *c* indicates the circle centre, and *R* is the radius of the circle in Scope Units. In implementing these difference equations in the Programme, the fullest possible use is made of the coordinate arithmetic capability of the TX-2 so that both the *x* and *y* equation computations are performed in parallel on 18 bit subwords. Even so, about ¾ of the total Sketchpad computation time is spent in line and circle generation. A vector and circle generating display would materially reduce the computational load of Sketchpad.

For computers which do only one addition at a time, the difference equations:

$$x_i = x_{i-1} + \frac{2}{R}(y_{i-1} - y_c)$$
$$y_i = y_{i-1} - \frac{2}{R}(x_i - x_c) \qquad (3)$$

should be used to generate circles. Equations (3) approximate a circle well enough and are known to close exactly both in theory and when implemented, because the *x* and *y* equations are dissimilar.

Digits and Text

Text, to put legends on a drawing, is displayed by means of special tables which indicate the locations of line and circle segments to make up the letters and numbers. Each piece of text appears as a single line of not more than 36 equally spaced characters which can be changed by typing. Digits to display the value of an indicated scalar at any position and in any size and rotation are formed from the same type face as text. It is possible to display up to five decimal digits with sign; binary to decimal conversion is provided, and leading zeros are suppressed.

Subpictures, whose use was seen in the introductory example above, are each represented in storage as a single *n*-component element. A subpicture is said to be an "instance" of its "master picture." To display an instance, all of the lines, text, *etc.* of its master picture must be shown in miniature on the display. The instance display Programme makes use of the line, circle, number, and text display Programmes and *itself* to expand the internal structure of the instance.

Display of Abstractions

The usual picture for human consumption displays only lines, circles, text, digits, and instances. However, certain very useful abstractions which give the drawing the properties desired by the user are represented in the ring structure storage. For example, the fact that the start and end points of a circle arc should be equidistant from the circle's centre point is represented in storage by a "constraint" block. To make it possible for a user to manipulate these abstractions, each abstraction must be able to be seen on the display if desired.

Not only does displaying abstractions make it possible for the human user to know that they exist, but also makes it possible for him to aim at them with the light pen and, for example, erase them. To avoid confusion, the display for particular types of objects may be turned on or off selectively by toggle switches. Thus, for example, one can turn on display of constraints as well as or instead of the lines and circles which are normally seen.

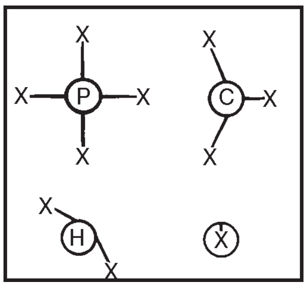


Figure : Display of Constraints.

If their selection toggle switch is on, constraints are displayed. The central circle and code letter are located at the average location of the variables constrained. The four arms of a constraint extend from the top, right side, bottom, and left side of the circle to the first, second, third, and fourth variables constrained, respectively. If fewer than four variables are constrained, excess arms are omitted. The constraints are shown applied to “dummy variables” each of which shows as an X.

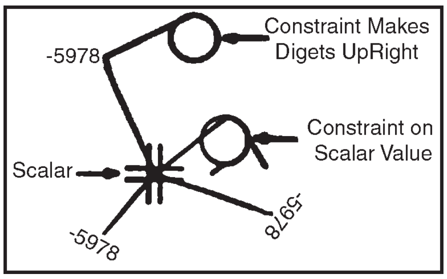


Figure : Three Sets of Digits Displaying the Same Scalar Value.

Another abstraction that can be displayed if desired is the value of a set of digits. For example, three sets of digits all displaying the same scalar value, -5978. The digits themselves may be moved, rotated, or changed in size, without changing the value displayed. If we wish to change the value, we point at its abstract display. The three sets of digits, all display the same value, as indicated by the lines connecting them to the #; changing this value would make all three sets of digits change. Constraints may be applied independently to either the position of the digits or their value as indicated by the two constraints in the figure.

Recursive Functions

In the process of making the Sketchpad system operate, a few very general functions were developed which make no reference at all to the specific types of entities on which they operate.

These general functions give the Sketchpad system the ability to operate on a wide range of problems. The motivation for making the functions as general as possible came from the desire to get as much result as possible from the programming effort involved. For example, the general function for expanding instances makes it possible for Sketchpad to handle any fixed geometry subpicture. The power obtained from the small set of generalized functions in Sketchpad is one of the most important results of the research.

In order of historical development, the recursive functions in use in the Sketchpad system are:

- Expansion of instances, making it possible to have subpictures within subpictures to as many levels as desired.
- Recursive deletion, whereby removal of certain picture parts will remove other picture parts in order to maintain consistency in the ring structure.

- Recursive merging, whereby combination of two similar picture parts forces combination of similarly related other picture parts, making possible application of complex definitions to an object picture.

Recursive Deleting

If a thing upon which other things depend is deleted, the dependent things must be deleted also. For example, if a point is to be deleted, all lines which terminate on the point must also be deleted. Otherwise, since, the *n*-component elements for lines contain no positional information, where would these lines end? Similarly, deletion of a variable requires deletion of all constraints on that variable; a constraint must have variables to act on.

Recursive Merging

If two things of the same type which are independent are merged, a single thing of that type results, and all things which depended on either of the merged things depend on the result of the merger.*

For example, if two points are merged, all lines which previously terminated on either point now terminate on the single resulting point. In Sketchpad, if a thing is being moved with the light pen and the termination flick of the pen is given while aiming at another thing of the same type, the two things will merge.

Thus, if one moves a point to another point and terminates, the points will merge, connecting all lines which formerly terminated on either. This makes it possible to draw closed polygons.

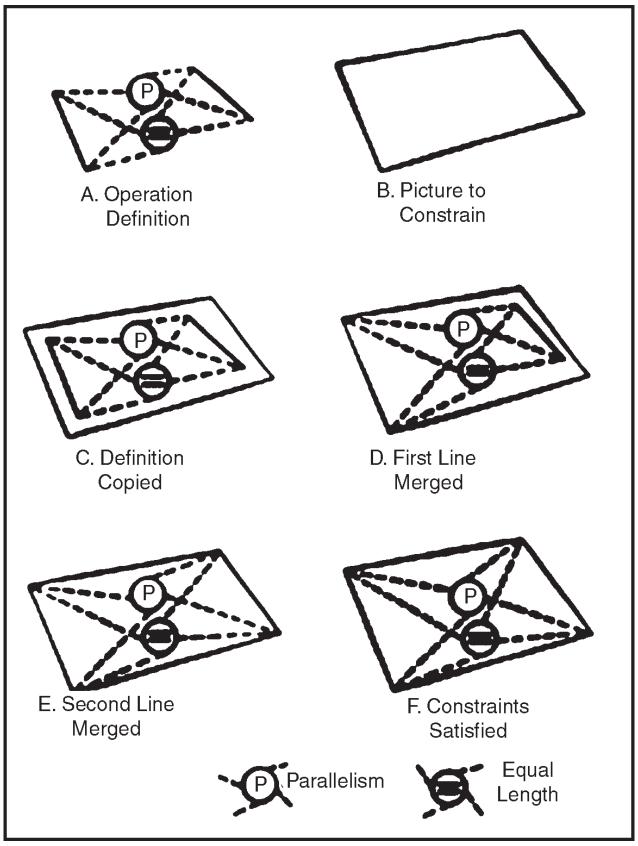


Figure : Applying a Two-constraint Definition to turn a Quadrilateral into a Parallelogram.

If two things of the same type which do depend on other things are merged, the things depended on by one will be forced to merge, respectively, with the things depended on by the other. The result of merging two dependent things depends, respectively, on the results* of the mergers it forces.* For example, if two lines are merged, the resultant line must refer to only two end points, the results of merging the pairs of end points of the original lines. All lines which terminated on any of the four original end points now terminate on the appropriate one of the remaining pair. More important and useful, all constraints which applied to any of the four original end points now apply to the appropriate one of the remaining pair. This makes it possible to speak of line segments as being parallel even though (because line segments contain no numerical information to be constrained) the parallelism constraint must apply to their end points and not to the line segments themselves. If we wish to make two lines both parallel and equal in length, the steps outlined. More obscure relationships between dependent things may be easily defined and applied. For example, constraint complexes can be defined to make line segments be collinear, to make a line be tangent to a circle, or to make the values represented by two sets of digits be equal. The "result" of a merger is a single thing of the same type as the merged things.

Recursive display of instances

The block of registers which represents an instance is remarkably small considering that it may generate a display of any complexity. For the purposes of display, the instance block makes reference to its master picture. The instance will appear on the display as a figure geometrically similar to its master picture at a location, size, and rotation indicated by the four numbers which constitute the "value" of the instance. The value of an instance is considered numerically as a four dimensional vector. The components of this vector are the coordinates of the centre of the instance and its actual size as it appears on the drawing times the sine and cosine of the rotation angle involved. In displaying an instance of a picture, reference is made to the master picture to find out what picture parts are to be shown. The master picture referred to may contain instances, however, requiring further reference, and so on until a picture is found which contains no instances. At each stage in the recursion, any picture parts displayed must be relocated so that they will appear at the correct position, size and rotation on the display. Thus, at each stage of the recursion, some transformation is applied to all picture parts before displaying them. If an instance is encountered, the transformation represented by its value must be adjoined to the existing transformation for display of parts within it. When the expansion of an instance within an instance is finished, the transformation must be restored for continuation at the higher level.

Attachers and Instances

Many symbols must be integrated into the rest of the drawing by attaching lines to the symbols at appropriate points, or by attaching the symbols directly to each other. For example, circuit symbols must be wired up, geometric patterns made by fitting shapes together, or mechanisms composed of links tied together appropriately. An instance may have any number of attachment points, and a point may serve as attacher for any number of instances. The light pen has the same affinity for the attachers of an instance that it has for the end point of a line.

An "instance-point" constraint, shown with code T, is used to relate an instance to each of its attachment points. An instance-point constraint is satisfied only when the point bears the same relationship to the instance that a master point in the master picture for that instance bears to the master picture coordinate system.

Any point may be an attacher of an instance, but the point must be designated as an attacher in the master drawing of the instance. For example, when one first draws a resistor, the ends of the resistor must be designated as attachers if wiring is to be attached to instances of it. At each level of building complex pictures, the attachers must be designated anew. Thus of the three attachers of a transistor it is possible to select one or two to be the attachers of a flip-flop.

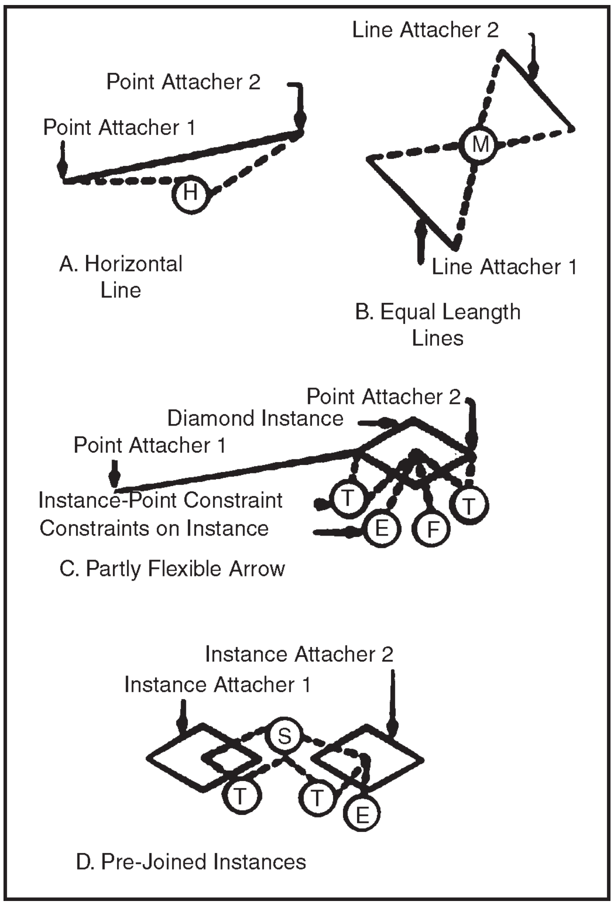


Figure : Definition Pictures to be Copied.

Building a Drawing, the Copy Function

At the start of the Sketchpad effort certain *ad hoc* drawing functions were programmed as the atomic operations of the system. Each such operation, controlled by a push button, creates in the ring structure a specific set of new drawing parts. For example, the "draw" button creates a line segment and two new end points (unless the light pen happens to be aimed at a point in which case only one new point need be created). Similarly, there are atomic operations for drawing circles, applying a horizontal or vertical constraint to the end points of a line aimed at, and for adding a "point-on-line" constraint whenever a point is moved onto a line and left there.

The atomic operations make it possible to create in the ring structure new picture components and relate them topologically. The atomic operations are, of course, limited to creating points, lines, circles, and two or three types of constraints. Since, implementation of the copy function it has become possible to create in the ring structure any predefined combination of picture parts and constraints at the press of a button. The recursive merging function makes it possible to relate the copied set of picture parts to any existing parts. For example, if a line segment and its two end points are copied into the object picture, the action of the "draw" button may be exactly duplicated in every respect. Along with the copied line, however, one might copy as well a constraint, Code H, to make the line horizontal, or two constraints to make the line both horizontal and three inches long, or any other variation one cares to put into the ring structure to be copied.

When one draws a definition picture to be copied, certain portions of it to be used in relating it to other object picture parts are designated as "attachers." Anything at all may be designated: for example, points, lines, circles, text, even constraints!.

The rules used for combining points when the "draw" button is pressed are generalized so that:

For copying a picture, the last-designated attacher is left moving with the light pen. The next-to-last-designated attacher is recursively merged with whatever object the pen is aimed at when the copying occurs, if that object is of like type. Previously designated attachers are recursively merged with previously designated object picture parts, if of like type, until either the supply of designated attachers or the supply of designated object picture parts is exhausted. The last-designated attacher may be recursively merged with any other object of like type when the termination flick is given. Normally only two designated attachers are used because it is hard to keep track of additional ones.

If the definition picture consists of two line segments, their four end points, and a constraint, Code M, on the points which makes the lines equal in length, with the two lines designated as attachers, copying enables the user to make any two lines equal in length. If the pen is aimed at a line when "copy" is pushed, the first of the two copied lines merges with it (taking its position and never actually being seen). The other copied line is left moving with the light pen and will merge with whatever other line the pen is aimed at when termination occurs. Since, merging is recursive, the copied equal-length constraint, Code M, will apply to the end points of the desired pair of object picture lines.

Copying Instances

The internal structure of an instance is entirely fixed. The internal structure of a copy, however, is entirely variable. An instance always retains its identity as a single part of the drawing; one can only delete an entire instance. Once a definition picture is copied, however, the copy loses all identity as a unit; individual parts of it may be deleted at will. One might expect that there was intermediate ground between the fixed-internal-structure instance and the loose-internal-structure copy. One might wish to produce a collection of picture parts, some of which were fixed internally and some of which were not. *The entire range of variation between the instance and the copy can be constructed by copying instances.*

For example, the arrow can be copied into an object picture to result in a fixed-internal-structure diamond arrowhead with a flexible tail. As the definition is set up, drawing diamond-arrowheaded lines is just like drawing ordinary lines. One aims the light pen where the tail is to end, presses "copy," and moves off with an arrowhead following the pen. The diamond arrowhead in this case will not rotate (constraint Code E), and will not change size (constraint Code F).

Copying pre-joined instances can produce vast numbers of joined instances very easily. For example, the definition, when repetitively copied, will result in a row of joined, equal size (constraint Code S) diamonds. In this case the instances themselves are attachers. Although each press of the "copy" button copies two new instances into the object picture, one of these is merged with the last instance in the growing row. In the final row, therefore, each instance carries all constraints which are applied to either of the instances in the definition. This is why only one of the instances carries the erect constraint, Code E.

Constraint Satisfaction

The major feature which distinguishes a Sketchpad drawing from a paper and pencil drawing is the user's ability to specify to Sketchpad mathematical conditions on already drawn parts of his drawing which will be automatically satisfied by the computer to make the drawing take the exact shape desired. The process of fixing up a drawing to meet new conditions applied to it after it is already partially complete is very much like the process a designer goes through in turning a basic idea into a finished design. As new requirements on the various parts of the design are thought of, small changes are made to the size or other properties of parts to meet the new conditions. By making Sketchpad able to find new values for variables which satisfy the conditions imposed, it is hoped that designers can be relieved of the need of much mathematical detail. The effort expended in making the definition of constraint types as general as possible was aimed at making design constraints as well as geometric constraints equally easy to add to the system.

Definition of a Constraint Type

Each constraint type is entered into the system as a generic block indicating the various properties of that particular constraint type. The generic block tells how many variables are constrained, which of these variables may be changed in order to satisfy the constraint, how many degrees of freedom are removed from the constrained variables, and a code letter for human reference to this constraint type.

The definition of what a constraint type does is a subroutine which will compute, for the existing values of the variables of a particular constraint of that type, the error introduced into the system by that particular constraint. For example, the defining subroutine for making points have the same x coordinate (to make a line between them vertical) computes the difference in their *x* coordinates. What could be simpler? The computed error is a scalar which the constraint satisfaction routine will attempt to reduce to zero by manipulation of the constrained variables. The computation of the error may be non--linear or time dependent, or it may involve parameters not a part of the drawing such as the setting of toggle switches, *etc.*

When the one pass method of satisfying constraints to be described later on fails, the Sketchpad system falls back on the reliable but slow method of relaxation to reduce the errors indicated by various computation subroutines to smaller and smaller values. For simple constructions such as the hexagon illustrated, the relaxation procedure is sufficiently fast to be useful. However, for complex systems of variables, especially directly connected instances, relaxation is unacceptably slow. Fortunately it is for just such directly connected instances that the one pass method shows the most striking success.

One Pass Method

Sketchpad can often find an order in which the variables of a drawing may be re-evaluated to completely satisfy all the conditions on them in just one pass. For the cases in which the one pass method works, it is far better than relaxation: it gives correct answers at once; relaxation may not give a correct solution in any finite time. Sketchpad can find an order in which to re-evaluate the variables of a drawing for most of the common geometric constructions. Ordering is also found easily for the mechanical linkages. Ordering cannot be found for the bridge truss problem.

The way in which the one pass method works is simple in principle and was easy to implement as soon as the nuances of the ring structure manipulations were understood. To visualize the one pass method, consider the variables of the drawing as places and the constraints relating variables as passages through which one might pass from one variable to another. Variables are adjacent to each other in the maze formed by the constraints if there is a single constraint which constrains them both. Variables are totally unrelated if there is no path through the constraints by which to pass from one to the other.

Suppose that some variable can be found which has so few constraints applying to it that it can be re-evaluated to completely satisfy all of them. Such a variable we shall call a "free" variable. As soon as a variable is recognized as free, the constraints which apply to it are removed from further consideration, because the free variable can be used to satisfy them. Removing these constraints, however, may make adjacent variables free. Recognition of these new variables as free removes further constraints from consideration and may make other adjacent variables free, and so on throughout the maze of constraints. The manner in which freedom spreads is much like the method used in Moore's algorithm to find the shortest path through a maze. Having found that a collection of variables is free, Sketchpad will re-evaluate them in reverse order, saving the first-found free variable until last. In re-evaluating any particular variable, Sketchpad uses only those constraints which were present when that variable was found to be free.

Examples and Conclusions

The library tape and thus serve to illustrate not only how the Sketchpad system can be used, but also how it actually has been used so far. We conclude from these examples that Sketchpad drawings can bring invaluable understanding to a user.

For drawings where motion of the drawing, or analysis of a drawn problem is of value to the user, Sketchpad excels. For highly repetitive drawings or drawings where accuracy is required, Sketchpad is sufficiently faster than conventional techniques to be worthwhile. For drawings which merely communicate with shops, it is probably better to use conventional paper and pencil.

Patterns

The instance facility enables one to draw any symbol and duplicate its appearance anywhere on an object drawing at the push of a button. This facility made the hexagonal pattern we saw. It took about one half hour to generate 900 hexagons, including the time taken to figure out how to do it. Plotting them takes about 25 minutes. The drafting department estimated it would take two days to produce a similar pattern.

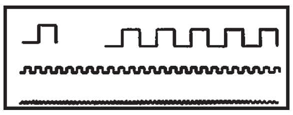


Figure : Zig-Zag for Delay Line.

The instance facility also made it easy to produce long lengths of the zig-zag pattern. A single "zig" was duplicated in multiples of five and three, *etc.* Five hundred zigs were generated in a single row. Four such rows were plotted one-half inch apart to be used for producing a printed circuit delay line. Total time taken was about 45 minutes for constructing the figure and about 15 minutes to plot it.

A somewhat less repetitive pattern to be used for encoding the time in a digital clock. Each cross in the figure marks the position of a hole. The holes are placed so that a binary coded decimal (BCD) number will indicate the time. Total time for placing crosses was 20 minutes, most of which was spent trying to interpret a pencil sketch of their positions.

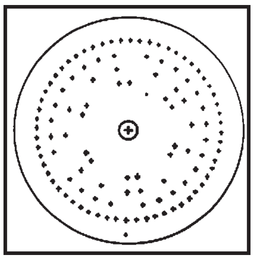


Figure : Binary Coded Decimal Encoder for Clock. Encoder was Plotted Exactly 12 Inches in Diameter for Direct use as a Layout.

Linkages

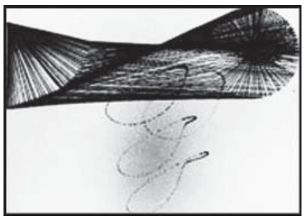


Figure : Three Bar Linkage. The Paths of Four Points on the Central Link are Traced. This is a 15 Second time Exposure of a Moving Sketchpad Drawing.

By far the most interesting application of Sketchpad so far has been drawing and moving linkages. The ability to draw and then move linkages opens up a new field of graphical manipulation that has never before been available. It is remarkable how even a simple linkage can generate complex motions. For example, only three moving parts. In this linkage a central " link is suspended between two links of different lengths. As the shorter link rotates, the longer one oscillates as can be seen in the multiple exposure. The motion of four points on the upright part of the “may be seen. To make the three bar linkage, an instance shaped like the “was drawn and given 6 attachers, two at its joints with the other links and four at the places whose paths were to be observed. Connecting the “ shaped subpicture onto a linkage composed of three lines with fixed length created the picture shown. The driving link was rotated by turning a knob below the scope. Total time to construct the linkage was less than 5 minutes, but over an hour was spent playing with it.

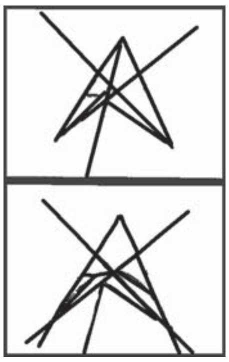


Figure : Conic Drawing Linkage. As the "Driving Lever" is Moved, the Point shown with a box Around it (in A) traces a Conic Section. This Conic can be Seen in the Time Exposure (B).

A linkage that would be difficult to build physically. This linkage is based on the complete quadrilateral. The three circled points and the two lines which extend out of the top of the picture to the right and left are fixed. Two moving lines are drawn from the lower circled points to the intersections of the long fixed lines with the driving lever. The intersection of these two moving lines (one must be extended) has a box around it. It can be shown theoretically that this linkage produces a conic section which passes through the place labeled “point on curve” and is tangent to the two lines marked “tangent.” A time exposure of the moving point in many positions. At first, this linkage was drawn and working in 15 minutes. Since, then we have rebuilt it time and again until now we can produce it from scratch in about 3 minutes.

Dimension Lines

To make it possible to have an absolute scale in drawings, a constraint is provided which forces the value displayed by a set of digits to indicate the distance between two points on the drawing.

This distance-indicating constraint is used to make the number in a dimension line correspond to its length. Putting in a dimension line is as easy as drawing any other line. One points to where one end is to be left, copies the definition of the dimension line by pressing the "copy" button, and then moves the light pen to where the other end of the dimension line is to be. The first dimension line took about 15 minutes to construct, but that need never be repeated since, it is a part of the library.

Bridges

One of the largest untapped fields for application of Sketchpad is as an input Programme for other computation Programmes. The ability to place lines and circles graphically, when coupled with the ability to get accurately computed results pictorially displayed, should bring about a revolution in computer application. By using Sketchpad's relaxation procedure we were to demonstrate analysis of the force distribution in the members of a pin connected truss.

A bridge is first drawn with enough constraints to make it geometrically accurate. These constraints are then deleted and each member is made to behave like a bridge beam. A bridge beam is constrained to maintain constant length, but any change in length is indicated by an associated number. Under the assumption that each bridge beam has a cross-sectional area proportional to its length, the numbers represent the forces in the beams.

The basic bridge beam definition (consisting of two constraints and a number) may be copied and applied to any desired line in a bridge picture by pointing to the line and pressing the "copy" button.

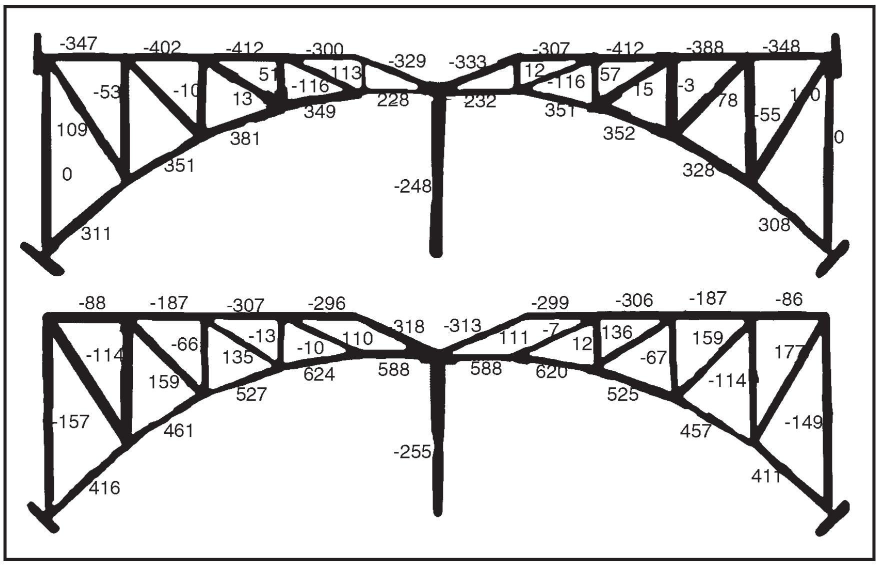


Figure : Cantilever and Arch Bridges. The Numbers Indicate the Forces in the Various Members as Computed by Sketchpad. Central load is not Exactly Vertical.

Having drawn a basic bridge shape, one can experiment with various loading conditions and supports effect of making minor modifications is.

For example, an arch bridge supported both as a three-hinged arch (two supports) and as a cantilever (four supports). For nearly identical loading conditions the distribution of forces is markedly different in these two cases.

Artistic Drawings

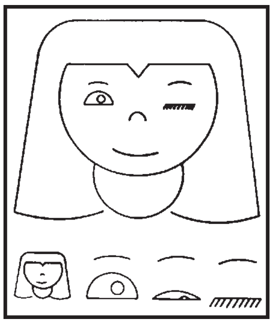


Figure : Winking girl, "Nefertite," and her Component Parts.

Sketchpad need not be applied exclusively to engineering drawings. For example, the girl "Nefertite" can be made to wink by changing which of the three types of eyes is placed in position on her otherwise eyeless face. In the same way that linkages can be made to move, a stick figure could be made to pedal a bicycle or Nefertite's hair could be made to swing. The ability to make moving drawings suggests that Sketchpad might be used for making animated cartoons.

Electrical Circuit Diagrams

Unfortunately, electrical circuits require a great many symbols which have not yet been drawn properly with Sketchpad and therefore are not in the library. After some time is spent working on the basic electrical symbols it may be easier to draw circuits. So far, however, circuit drawing has proven difficult.

The circuits are parts of an analog switching scheme. You can see in the figure that the more complicated circuits are made up of simpler symbols and circuits. It is very difficult, however, to plan far enough ahead to know what composites of circuit symbols will be useful as subpictures of the final circuit. The simple circuits were compounded into a big circuit involving about 40 transistors. Including much trial and error, the time taken by a new user (for the big circuit not shown) was ten hours. At the end of that time the circuit was still not complete in every detail and he decided it would be better to draw it by hand after all.

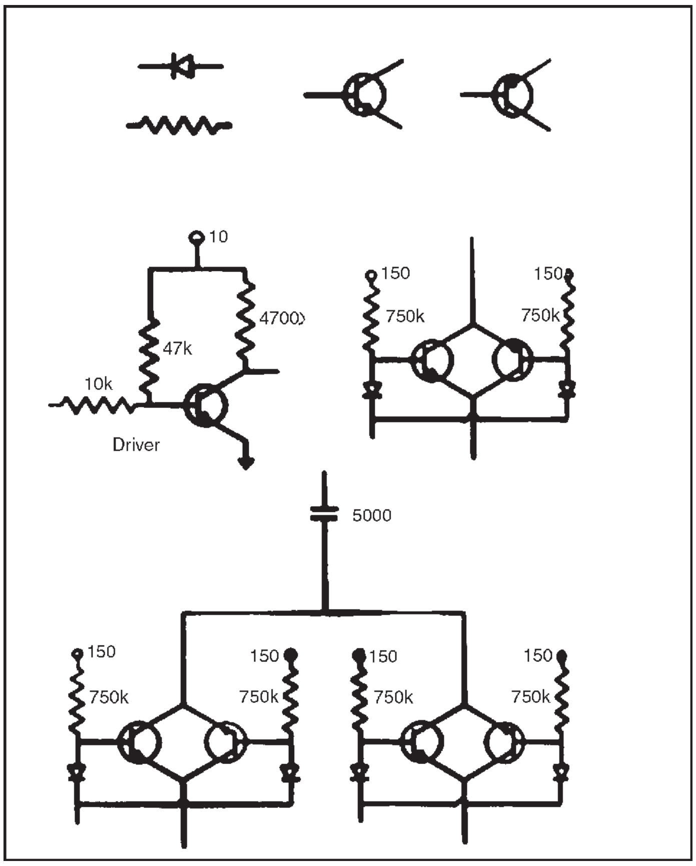


Figure : Circuit Diagram. These are parts of the Large Circuit Mentioned in the Text.

Conclusions

The circuit experience points out the most important fact about Sketchpad drawings. It is only worthwhile to make drawings on the computer if you get something more out of the drawing than just a drawing. In the repetitive patterns we saw in the first examples, precision and ease of constructing great numbers of parts were valuable. In the linkage examples, we were able to gain an understanding of the behaviour of a linkage as well as its appearance. In the bridge examples we got design answers which were worth far more than the computer time put into them. If we had a circuit simulation Programme connected to Sketchpad so that we would have known whether the circuit we drew worked, it would have been worth our while to use the computer to draw it. We are as yet a long way from being able to produce routine drawings economically with the computer.

Future Work

The methods outlined in this section generalize nicely to three dimensional drawing. In fact, the work reported in "Sketchpad III" by Timothy Johnson will let the user communicate solid objects to the computer. Johnson is completely bypassing the problem of converting several two dimensional drawings into a three dimensional shape. Drawing will be directly in three dimensions from the start. No two dimensional representation will ever be stored. Work is also proceeding in direct conversion of photographs into line drawings. Roberts reports a computer program able to recognize simple objects in photographs well enough to produce three dimensional line drawings for them. Roberts is storing his drawings in the ring structure described here so that his results will be compatible with the three dimensional version of Sketchpad.

Major improvements to Sketchpad of the same order and power as the existing definition copying capability can be foreseen. At present Sketchpad is able to add defined relationships to an existing object drawing. A method should be devised for defining and applying changes which involve removing some parts of the object drawing as well as adding new ones. Such a capability would permit one to define, for example, what rounding off a corner means. Then, one could round off any corner by pointing to it and applying the definition.

3

Web-based Multimedia

The evolution of the Web is sometimes described in terms of first, second and third generation Web content. In the first generation, the Web browser provided its users a uniform interface to a wide variety of information on the Internet. URIs provide a simple but universal naming scheme, and HTTP a simple but fast transfer protocol. In theory, HTML was designed to provide the "glue" between various information resources in the form of hyperlinks, and as a default document format Web servers could resort to when other available formats were not understood by the client. In practice, however, HTML turned out as being the format that was also used to put the bulk of the content on the Web. A major problem of the first generation Web content was the fact that all this HTML content was manually written. This proved to be too inflexible when dealing with content that is stored in existing databases or that is subject to frequent changes. For larger quantities of handwritten documents, keeping up with changing browser technology or updating the “look and feel” proved to be hard. In the current, second generation Web, the required flexibility is provided by a range of technologies based on automatic generation of HTML content.

Approaches vary from filling in HTML templates with content from a database back-end to applying CSS and XSLT style sheets to give the content the appropriate look and feel while storing the content itself in a form free of presentation and browser related details.

Current trends on the Web make the flexibility provided by the second generation Web technology even more relevant. The PC-based Web browser is no longer the only device used to access the Web.

Content providers need to continuously adapt their content to take into account the significant differences among Web access using PCs and alternative devices ranging from small-screen mobile phones and hand-held computers to set-top boxes for wide-screen, interactive TV. Additional flexibility is required to take into account the different languages, cultural backgrounds, skills and abilities of the wide variety of users that may access their content.

By providing flexibility in terms of the presentation and user interaction, second generation Web technology primarily addresses the needs of human readers. In contrast, third generation Web technology focuses on content that is both human *and* machine processable. Machine-processable content is a pre-requisite for the more intelligent services that constitute the “Semantic Web” as envisioned by Tim Berners-Lee and others.

To provide real machine-processable content, next generation Web technology primarily needs to extend interoperability on the semantic level. Current Web recommendations focus on either syntactic issues or on semantics for a generic domain. Examples of such generic domains that are covered by current W3C specifications include the semantics of presentation (CSS, XSL), interaction (XLink, XForms), privacy (P3P) and content rating (PICS). The third generation, however, needs to provide interoperability in terms of application and domain-dependent semantics. A first step in this direction has already been taken by W3C specifications such as XML and RDF.

New models and tools to improve the support for second and third generation Web currently receive ample attention, both in research and commercial environments. Most of this attention, however, is directed towards text-oriented applications. *Multimedia* content — that is, content that seamlessly integrates multiple media types in a synchronized, interactive presentation — has some characteristics that are fundamentally different from text. These differences mean that the models and tools that are developed for text cannot be readily applied to multimedia. In this chapter we claim that — while the need for second and third generation multimedia content is similar to that for textual content — the technical requirements to support this need are substantially different.

The structure of the remainder of the article is as follows. First, we analyse the requirements for multimedia presentation generation, focusing on the differences between text and multimedia document transformations. Then we describe the different levels of abstraction that characterize the *Cuypers* multimedia presentation generation system (the system is named after the Dutch architect who designed several famous buildings in Amsterdam, including the Rijksmuseum and Central Station). We discuss the use of these abstraction levels in the context of an example scenario. We conclude with an overview of related work and a description of future work.

Requirements for Second Generation Multimedia Content

Many of the advantages of generated Web content over manually authored Web content are commonly known and well described in the research literature. When the content and its underlying structure are stored separately from the details of a specific presentation of that content, tools can be developed to automatically adapt the presentation to the current situation, both in terms of the capabilities of the technical infrastructure and the specific needs of the user. These advantages not only apply to text, but perhaps even more to multimedia.

One can argue that adaptation to the available network bandwidth, presentation capabilities of the end-user's platform, preferred language and preferred media types is even more important for complex, interactive multimedia than for content that is mainly text-based. In this chapter, we explain the differences between the requirements for second generation text and multimedia content.

Standards such as SGML [SGML:ISO] and XML [XML1:W3C] use embedded markup to encode documents in a presentation-independent way in order to increase longevity, reusability and flexibility. The visual appearance of these so-called *structured documents* is defined by the specification of a *style sheet*. Style sheets effectively define a mapping between the abstractions in the document structure and those in the presentation structure.

This mapping is defined using a style sheet language. In general, efficient run-time execution and standardization of the style language (in order to be able to interchange documents) is considered more important than the language's flexibility and expressiveness. Additionally, for most applications, the mapping can be described, relatively straightforwardly, in a functional way — standardized style and transformation languages such as DSSSL [DSSSL:ISO] and XSLT [XSLT:W3C] are functional languages.

Style sheets as described above mainly come in two flavours: template-driven and content-driven. Template-driven style sheets first set up the designed layout, and then fill in the content by querying the source document or underlying database. This works well when (a) the underlying structure of the content is sufficiently known to allow effective querying, (b) the structure of the generated presentations is comparable and known in advance and (c) the presentation structure is independent of the results returned by the query. For example, the type and size of the information returned by each query and the number of items queried for are required to be known in advance.

In contrast, in content-driven style sheets the size and global structure of the generated presentation is not known in advance, so it cannot be defined by a template specification. Instead, the style sheet defines a set of transformation rules that will be applied to the source document. Again, certain aspects of the structure of the underlying content should be known, this time not to allow querying, but to allow the definition of an effective *selector* which determines to which element(s) each rule applies. In content-driven style sheets, the structure and size of the generated presentation varies largely with the structure and size of the underlying source document. For most textual documents, this is not a problem, since most textual elements can be flexibly nested and chained. Strict constraints on the size of the resulting presentations are also rare: for online presentations, scrollbars make the page length irrelevant, and for paged-media, extra pages can always be added.

Today's style languages are, however, not suited for those rare cases that size does matter. It is, for example, extremely hard (if not impossible) to write a style sheet that formats this HTML paper and makes sure it exactly meets the conference's 10-page limit. For many text-based applications, however, these constraints do not apply, and the techniques described above work well. For almost all multimedia applications, however, spatio-temporal positioning is not this flexible. In addition, there are a number of other issues that prevent the use of text-oriented techniques for multimedia document generation:

Multimedia uses Different Document and Presentation Abstractions

The separation of the document's structure from its visual appearance is a fundamental and well known abstraction technique, both in database and structured document technology. A less common distinction is that between the specification of the document's visual appearance and its realization in terms of the final-form presentation format. For example, style languages such as XSL (and also DSSSL) define an abstract formatting object model that can be used to define the visual appearance of a presentation in a way that is independent of format-specific details of the final-form presentation. In this way, a single XSL style sheet can be defined for a specific set of documents, and, depending on the available back-ends for the XSL formatting model, the same style sheet can be applied for the generation of an online, PostScript or RTF version.

XSL style sheets essentially map document abstractions onto presentation abstractions. For text, this works well because on both levels, we have a commonly established set of abstractions to work with. On the document level, chapters, sections, headings and titles, bulleted lists, etc., are abstractions that are frequently used across different tools. On the presentation level, the same applies to abstractions such as pages, columns, inline, block-level and floating material, font families, etc. These abstractions are not only applicable across many domains, they have also proven to be extremely stable over time: while the majority of these abstractions originate from the early days of printing, they are still highly applicable to today's online documents. Even for text, however, transformations based on the abstract formatting models of XSL or DSSSL are not (yet) widely used: most tools still transform directly into target formats such as HTML or WML, by-passing the abstract formatting model entirely. While this may be partly because of the relative newness of the XSL specification, another reason is that the advantages in terms of reuse do not always outweigh the disadvantages in terms of decreased flexibility and increased complexity.

For multimedia, we still lack a commonly accepted set of abstractions, both on the document and the presentation level. The relatively slow acceptance of abstract formatting models for text, combined with rapidly changing multimedia technology and the vast range of different multimedia applications and presentation features, will make it very unlikely that this situation is going to change in the near future. This is highly unfortunate, because it means that a major advantage of today's style sheet technology — the definition of style sheets independent from the syntactic details of the target presentation format — cannot be applied to multimedia.

Multimedia Document Formatting

For text, we have an established set of (complicated but) well understood algorithms that can be used to automatically typeset a text according to the requirements of a given layout specification. To keep the style sheet itself as declarative as possible, the components implementing these relatively low level and detailed algorithms are typically part of the style engine's back-end application. These back-end components typically implement kerning, hyphenation, justification, and line and page breaking algorithms. Note that these algorithms are based on the linear structure of the underlying text. Since multimedia documents are not based on such a text flow, these algorithms do not suffice for formatting multimedia documents. For example, in text-based formatting, content that does not fit on a single page or screen is just spread out over multiple pages, or rendered on a scrollable area that is larger than the screen. These solutions are, in general, not applicable to multimedia presentations, where the very concept of a page or scrollbar often does not make sense.

In addition, many document-level and presentation-level abstractions for text are also based on text flow. For example, in a style language such as CSS, a key concept such as *relative positioning* refers to the ability to specify the position of an object relative to its default position in the text flow. Such flow-based models are, in general, not applicable to multimedia documents.

Multimedia Transformations

Most style and transformation languages do not support feedback from the rendering application back into the main transformation process. For example, information about the precise position onto which a specific word is rendered, is only available after the rendering application has fully formatted the document. Consequently this type of information is not available in the transformation process. For text, this limitation hardly ever causes problems: due to the flexibility of the text flow, the system can in most cases adjust the layout to make it meet the given constraints. For multimedia, however, the only way to determine whether a given layout specification can be realised with respect to a given set of constraints is to actually solve this set of constraints. This task is typically performed not by a (high level) transformation engine, but by a (lower level) constraint solver implemented in the back-end. For multimedia, it is thus of crucial importance to allow feedback from the lower levels of the process to the higher levels.

Functional Model of Multimedia Transformations

Transforming a presentation-independent structure to a presentation of acceptable quality is, when compared with text-centric presentations, relatively complex for media-centric presentations. These mappings are often best specified using a trial and error strategy, by backtracking over a set of alternative presentation rules, trying out different sets of constraints along the way. In contrast, most textual transformations are relatively straightforward mappings that can be better specified in a set of functional style rules. The more complex transformations that are common in multimedia are more conveniently expressed in a logic-based language with built-in support for backtracking and constraint solving.

Levels of Abstraction in Cuypers

Cuypers is a research prototype system, developed to experiment with the generation of Web-based presentations as an interface to semi-structured multimedia databases. Cuypers addresses many of the issues discussed in the previous section. First of all, it explores a set of abstractions, both on the document and on the presentation level, that are geared towards interactive, time-based and media centric presentations, rather than presentations that are based on text-flow. Second, it uses a set of easily extensible transformation rules specified in Prolog, exploiting Prolog's built-in support for backtracking. Finally, it facilitates easy feedback between the higher and lower level parts of the transformation process by executing both within the same environment. Instead of a strict separation between the transformation engine and the constraint solver, our system uses a constraint solver embedded in Prolog, so the system is able to backtrack when the transformation process generates a set of insolvable constraints.

Cuypers operates in the context of the environment depicted in Figure below. It assumes a server-side environment containing a multimedia database management system, an intelligent multimedia IR retrieval system, the Cuypers generation engine itself, an off-the-shelf HTTP server, and — optionally — an off-the-shelf streaming media server. At the client-side, a standard Web client suffices. The focus of this chapter is the Cuypers generation engine. Given a rhetorical (or other type of semantic) structure and a set of design rules, the system generates a presentation that adheres to the limitations of the target platform and supports the user's preferences.

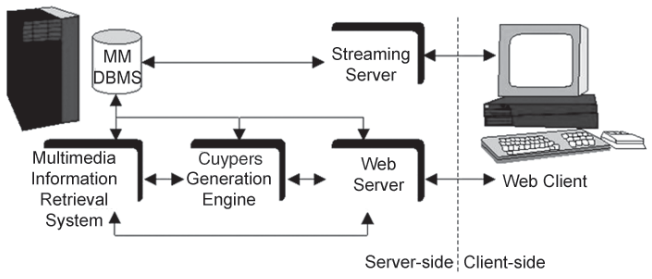


Fig. The Environment of the Cuypers Generation Engine

The experience gained from the development of earlier prototypes (*e.g.* work done by Bailey) however, proved that for most applications, the conceptual gap between an abstract, presentation-independent document structure and a fully-fledged, final-form multimedia presentation is too big to be specified by a single, direct transformation. Instead, we take an incremental approach, and define the total transformation in terms of smaller steps, which each perform a transformation to another level of abstraction. These levels are depicted in Figure below and include the semantic, communicative device, qualitative constraint, quantitative constraint and final-form presentation levels, resp.

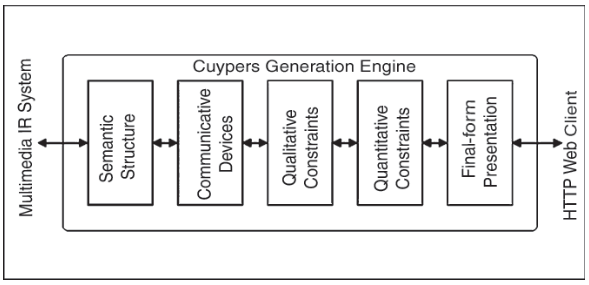


Fig. The Layers of the Cuypers Generation Engine

We describe each abstraction level and why it is needed in the overall process. We take a bottom-up approach and start with the final-form presentation level, which is the level that describes the presentation as it is delivered to the client's browser. This is also the level readers will be most familiar with, since it describes documents on the level of their encoding in for example HTML [XHTML10:W3C], SMIL [SMIL20:W3C] or SVG[SVG:W3C]. We subsequently add more abstraction levels, and end with the highest level, the "semantic level", which completely abstracts from all layout and style related information.

Final-form Presentation Level

At the lowest level of abstraction, we define the final-form presentation, which encodes the presentation in a document format that is readily playable by the end user's Web browser or media player. Examples of such formats include, HTML, SVG, and — the focus of our current prototype — SMIL. This level is needed to make sure that the end-user's Web-client remains independent of the abstractions we use internally in the Cuypers system, and to make sure that the end-user can use off-the-shelf Web clients to view the presentations generated by our system.

Quantitative Constraints Level

To be able to generate presentations of the same information using different document formats, we need to abstract from the final-form presentation. On this level of abstraction, the desired temporal and spatial layout of the presentation is specified by a set of format-independent constraints, from which the final-form layout can be derived automatically.

An example of a quantitative constraint is "the x-coordinate of the top-right corner of picture X should be at least 10 pixels smaller than, than the x-coordinate of the top-left corner of picture Y". Such constraints provide a first level of abstraction, abstracting from the syntactic details of the final-form presentation format, but also from the presentation's exact numeric layout specifications.

While more abstract than the final form presentation, a specification at this level provides sufficient information for the Cuypers system to be able to automatically generate the final-form presentation. An off-the-shelf numeric constraint solver is used to determine whether or not a given layout specification can be realised, and, if so, to generate any numeric layout specifications needed. The use of constraints also gives the system the flexibility to automatically adapt to small differences in screen size between, for example, two different handhelds or mobile phones.

In practice, larger differences cannot be solved at this level of abstraction. The use of numeric constraints is, for example, not sufficient to cater for the differences between the small screen of a mobile phone versus the large screen of a desktop web browser. Another drawback is that it is hard to specify higher level requirements such as the fact that certain rules should be applied consistently across the entire layout. In addition, for some final-form formats this level of abstraction is just too low to be useful. For example, for the relatively flat spatial layout of a SMIL 1.0 document, the constraints given above are well-suited. The same constraints, however, are too low-level to generate the complex temporal hierarchy that gives a SMIL presentation its adaptive scheduling information. On the implementation level, numeric constraints also have serious drawbacks. For example, when automatic backtracking over alternative layouts is used, a set of quantitative constraints might generate alternative layouts which are all equal, except for one coordinate, whose value only increases or decreases with one pixel (or other unit) for each generated layout.

The discussion above can be summarized by stating that numeric or quantitative constraints are necessary because solving a set of quantitative constraints is the only way to determine whether a specific layout can be realised with respect to a specific requirements. In addition, many final-form formats use numeric information to define the layout presentation. For many other purposes, however, these constraints are too low level and contain too much detail. Qualitative constraints are introduced to solve these problems.

Qualitative Constraints Level

An example of a qualitative constraint is "caption X is positioned below picture Y", and backtracking to produce alternatives might involve trying right or above, etc. Some final-form formats allow specification of the document on this level. In these cases, the Cuypers system only generates and solves the associated numeric constraints to check whether the presentation can be realised at all, it subsequently discards the solution of the constraint solver and uses the qualitative constraints directly to generate the final form output.

In the Cuypers system, qualitative constraints also provide a basis for defining *meta-constraints*. Meta-constraints are necessary to specify more global properties of the resulting document, and are used with Cuypers to ensure consistency across the presentation. For example, to prevent some captions from appearing, a designer could add a meta-constraint specifying that all captions should appear. Meta-constraints derive their name from the fact that they are implemented as constraints that constrain the set of generated constraints. While qualitative constraints solve many of the problems associated with quantitative constraints, they are still not suited for dealing with the relatively large differences in layout, *e.g.*, as in the mobile phone versus the desktop browser example given above. Therefore, another level of abstraction is introduced: the communicative device.

Communicative Device Level

The highest level of abstraction describing the presentation's layout makes use of *communicative devices*.

These are similar to the patterns of multimedia and hypermedia interface design described by in that they describe the presentation in terms of well known spatial, temporal and hyperlink presentation constructs. An example of a communicative device described in is the *bookshelf*. This device can be effectively used in multimedia presentations to present a sequence of media items, especially when it is important to communicate the *order* of the media items in the sequence.

How the bookshelf determines the precise layout of a given presentation in terms of lower level constraints can depend on a number of issues. For example, depending on the cultural background of the user, it may order a sequence of images from left to right, top to bottom or *vice-versa*. Also its *overflow strategy*, that is, what to do if there are too many images to fit on the screen, may depend on the preferences of the user and/or author of the document. It may decide to add a "More info" hyperlink to the remaining content in HTML, alternatively, it could split the presentation up in multiple scenes that are sequentially scheduled over time in SMIL.

Note that communicative devices,, typically deal with layout strategies that involve multiple dimensions (including space, time and linking), while the constraints typically do not cross the boundaries of a single dimension. Constraints using variables along more than one dimension are called cross-dimensional constraints, and have previously been discussed in. The introduction of such constraints would simplify the definition of many communicative devices and is the subject of further research.

While the communicative device level is a very high-level description of the presentation, we still need a bridge from the domain-dependent semantics as stored in the multimedia information retrieval system to the high-level hypermedia presentation devices. To solve this problem, we introduce one last level of abstraction: the semantic structure level.

Semantic Structure Level

This level completely abstracts from the presentation's layout and hyperlink navigation structure and describes the presentation purely in terms of higher-level, "semantic" relationships. In the current Cuypers system we focus on the rhetorical aspects of the presentation, because it applies well to the application domains for which we are currently building prototypes (*e.g.* generating multimedia descriptions of artwork for educational purposes).

Depending on the target application, however, other types of semantic relations can be used as well. Possible alternatives include abstractions based on the presentation's narrative structure for story-telling applications or abstractions based on an explicit discourse model.

From the perspective of the Cuypers architecture, any set of semantic relations can be chosen as long as it meets the following two requirements:

- It should sufficiently abstract from all presentation details so that these can be adequately adapted by the lower levels of the system, and
- It should provide sufficient information so that the relations can be used to generate an adequate set of communicative devices that convey the intended semantics to the end user.

The subdivision of the generation process in Cuypers is based on these different levels, with an extensible set of transformation rules to move from one level to another.

In practice, however, the transformations work by backtracking up and down different levels, and transformation rules may have access to information from several steps earlier. To explain the different abstraction levels and the associated transformation steps.

Example Scenario

We use an example scenario where the user (studying art history) just asked the system to explain the use of the *chiaroscuro* technique (strong contrast of light and dark shading) in the paintings of Rembrandt van Rijn.

The system's multimedia information retrieval back-end queried its annotated multimedia database system and retrieved five images of paintings that are annotated as using this technique, the accompanying titles and a general textual description of the term *chiaroscuro*.

Semantic Level: Rhetorical Structure

A presentation is constructed around the concept "Examples of Chiaroscuro in the works of Rembrandt van Rijn", using the images as *examples* of the core concept, and the text as an *elaboration* of the core concept.

Additionally, to preserve the ordering of the time the picture was made, the five images are shown in a *sequence* relation. The resulting

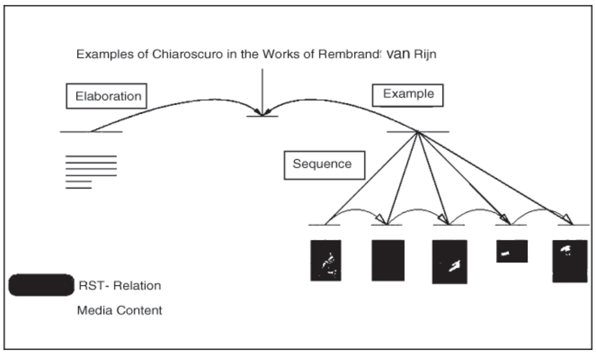


Fig. RST Tree Representation of the Input

tree structure, using the notation common in Rhetorical Structure Theory. The tree is encoded using a simple XML Schema to represent RST nucleus/satellite relations.

Note that we do *not* expect content authors to directly use the semantic abstractions nor the rhetoric markup during the authoring phase. Automatic generation of these structures, however, requires advanced knowledge of the domain, the organization of the multimedia database, the users and their task, and is the topic of future research. In the current Cuypers prototype, the generation of the rhetorical structure is simply hardwired into the server's multimedia information retrieval system, which is considered to be beyond the scope of this chapter. Here, we focus on the Cuypers presentation engine, and assume the RST structure as the input given to the engine.

```
        <!DOCTYPE presentation PUBLIC "-//CWI/DTD Rhetorics
1.0//EN" "rhetoric.dtd">
           <presentation xmlns="http://www.cwi.nl/~media/ns/
cuypers/rhetoric">
               <media id="title"... refs to content/metadata
database .../>
               <media id="img1" ... />
               <media id="img2" ... />
               <media id="img3" ... />
               <media id="chiaroscuro" ... />
```

```
    <nsRelation>
      <nucleus>
        <mediaref idref=”title”/>
      </nucleus>
      <satellite type=”example”>
        <mnRelation type=”sequence”>
          <nucleus>
            <mediaref idref=”img1"/>
          </nucleus>
          ... ...
          <nucleus>
            <mediaref idref=”img5"/>
          </nucleus>
        </mnRelation>
      </satellite>
      <satellite type=”elaboration”>
       <mediaref idref=”chiaroscuro”/>
      </satellite>
    </nsRelation>
  </presentation>
```

Fig. ***XML Encoding of the Presentation's Rhetorical Structure***

High-level Presentation Semantics

Note that the rhetorical structure contains no information about the spatio-temporal layout of the final-form presentation. This information is incrementally added by the Cuypers system, based on general design knowledge, combined with knowledge about the underlying domain (*e.g.* “17th century painting”), the task and preferences of the end-user and the capabilities of the device that is used to access the Web.

In the first step, the input is matched against a set of rules designed to convert the input to a communicative device hierarchy. Note that this is purely a design decision: in practice, designers of a particular application will need to extend the default rule set that comes with the Cuypers system.

The rules that match the input RST structure could, for example, map the root nucleus (the label "Chiaroscuro by Rembrandt van Rijn") to the title of the presentation. In addition, the rules determine that the title, elaborative text and the example section should be visible at the same time, as close to another as possible.

This is used by grouping these elements in a communicative device named *spatial adjacency*. Because the example section itself consisted of a sequence of images of which the ordering should be preserved, the sequence is mapped to a communicative device named *bookshelf*. The bookshelf's layout strategy is parameterized, in this case the strategy is to try to achieve a left-to-right, top-to-bottom ordering first, and to use a temporal overflow strategy when it proves impossible to fit all images on a single screen.

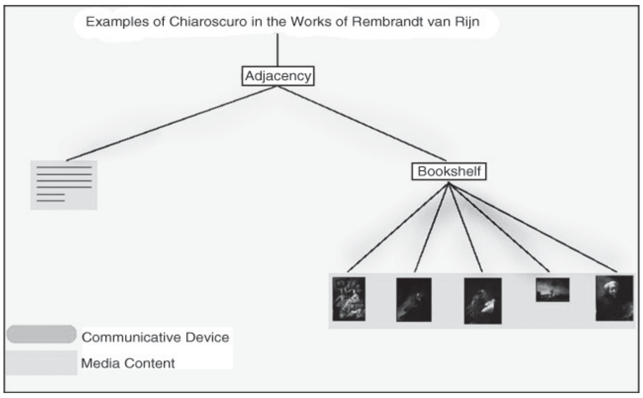


Fig. Example Communicative Device Hierarchy

Qualitative Constraint Level

While the communicative device hierarchy, reflects the most basic design decisions about the way the document should be communicated to the user, the mutual relationships among the media items have not been established. This is done in the qualitative constraint level, which converts the communicative device hierarchy to a graph structure.

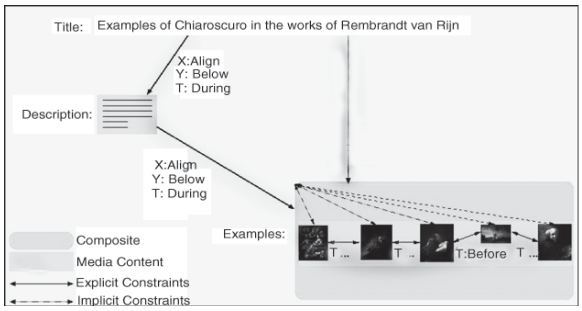


Fig. Example Qualitative Constraint Graph

The graph structure consists of nodes and edges, where the nodes represent the media items and the edges between the nodes are labelled with the constraints that relate them. *Composite* nodes can be used to model useful hierarchical relationships between media items at the constraint level. The resulting graph after backtracking over several alternative solutions for converting the communicative device structure. In this case, it turned out that the user's screen size is too small to display more than one painting at a time. As a result, all alternatives that tried a left-to-right, top-to-bottom ordering of the paintings failed, and the bookshelf has resorted to its overflow strategy: it decides to show the paintings one after the other, sequentially ordered in time. During the time the images are shown, it makes sure that the title and descriptive text. Also note that to define the communicative devices in terms of qualitative constraints, only a limited number of constraints need to be specified directly. Most constraints can be automatically generated by the system. For example, if the title is to be displayed during the description, and the description is to be displayed during the examples, the system automatically derives that the title is to be displayed during the examples. These automatically generated constraints are used when checking consistency rules such as "always show a title when displaying something else". In this case, the system knows that the title is shown during the examples, even when this is not explicitly specified by the transformation rules.

Quantitative Constraint Level

To be able to check whether a proposed multimedia layout can be realised, all the constraints need to be resolved on the lowest level.

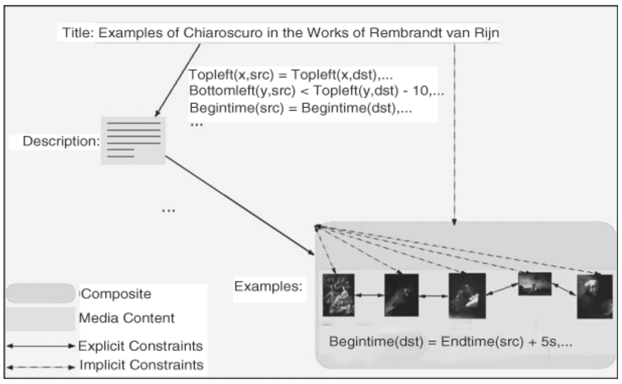

Title: Examples of Chiaroscuro in the Works of Rembrandt van Rijn
Topleft(x,src) = Topleft(x,dst),...
Bottomleft(y,src) < Topleft(y,dst) - 10,...
Begintime(src) = Begintime(dst),...
...
Description:
...
Examples:
Begintime(dst) = Endtime(src) + 5s,...
Composite
Media Content
Explicit Constraints
Implicit Constraints

Fig. Example (Partial) Quantitative Constraint Graph

For the spatial and temporal dimensions, this means that all the qualitative constraints need to be converted into numeric or quantitative constraints.

So if three images of a certain size are to be positioned left of one another with a certain minimum padding, at this point the system needs to do the associated mathematics to find out whether and how this can be done: it reformulates all the qualitative constraints into numerical constraints, fills in the actual sizes of the images and acceptable padding distances, and tries to solve the given set of constraints. The conversion process involves many quantitative constraints that are automatically generated.

This is, however, very efficient from an implementation point of view: the more constraints that are added, the smaller the constraint variable domains become, and the faster a solution will be generated.

Note that part of the information generated at this step is only needed to make sure that layouts meet the given constraints.

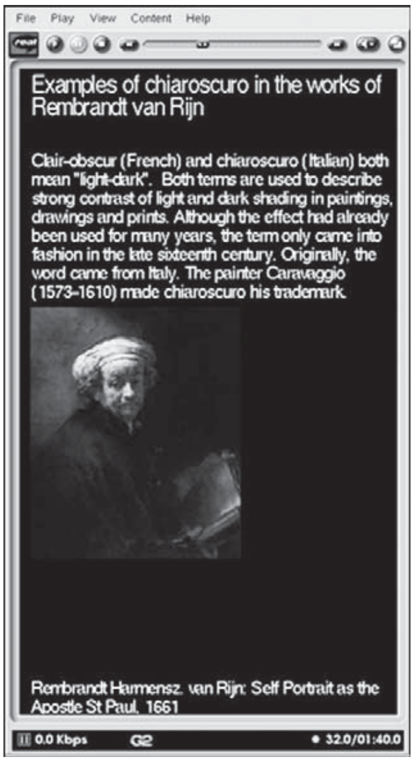
Parts of the solution itself are too low-level to be useful in high-level formats. Other parts of the solution, however are directly used and copied almost verbatim into the encoding of the final-form presentation.

Final-form Generation

In the last step, the information accumulated in the previous steps is used to generate the final presentation in SMIL. A snapshot of the result is shown in Figure.

The resulting SMIL markup is listed. As one can see, the encoding used for the layout specification in the head is rather low level, and these are indeed the direct values generated by solving the set of quantitative constraints.

In contrast, the temporal hierarchy in the body has been generated on the basis of the qualitative (Allen) constraints, realizing during constraints with <par> elements in SMIL, and after constraints with<seq> elements.



Examples of chiaroscuro in the works of Rembrandt van Rijn

Clair-obscur (French) and chiaroscuro (Italian) both mean "light-dark". Both terms are used to describe strong contrast of light and dark shading in paintings, drawings and prints. Although the effect had already been used for many years, the term only came into fashion in the late sixteenth century. Originally, the word came from Italy. The painter Caravaggio (1573-1610) made chiaroscuro his trademark.

Rembrandt Harmensz. van Rijn: Self Portrait as the Apostle St Paul. 1661

Fig. Snapshot of the Resulting SMIL Presentation (RealPlayer).

```
      <?xml version="1.0"?>
      <!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 1.0//EN"
       "http://www.w3.org/TR/REC-smil/SMIL10.dtd">
       <smil>
        <head>
          <meta name="generator" content="Cuypers 1.0"/>
          <layout>
            <root-layout id="root-layout" background-
colour="black"
                width="400" height="690"/>
            <region id="title" left="10" top="5"
               width="400" height="50" fit="meet"/>
            <region id="descr" left="10" top="55"
                width="400" height="200" fit="meet"/>
            <region id="img" left="10" top="255"
                width="400" height="400" fit="meet"/>
            <region id="ptitle" left="10" top="655"
                width="400" height="35" fit="meet"/>
          </layout>
        </head>
```

```
         <body>
          <par>
           <text  region=”title”  src=”...query  to  multimedia
database...”/>
             <text  region=”descr”  src=”...”/>
            <seq>
             <par  dur=”10">  ...  1st  painting+title  ...  </
par>
             <par  dur=”10">  ...  2nd  painting+title  ...  </
par>
             <par  dur=”10">  ...  3rd  painting+title  ...  </
par>
             <par  dur=”10">  ...  4th  painting+title  ...  </
par>
              <par  dur=”10">
                  <img  region=”img”  src=”...”/>
                 <text  region=”ptitle”  src=”..”/>
               </par>
             </seq>
            </par>
           </body>
          </smil>
```

Fig. SMIL Encoding of the Presentation Shown in Figure Above.

Implementation of Multimedia Presentation

To exploit the possibilities offered by on demand multimedia presentation integration, we have integrated the Cuypers core presentation generation engine with an off-the-shelf HTTP server (Apache).

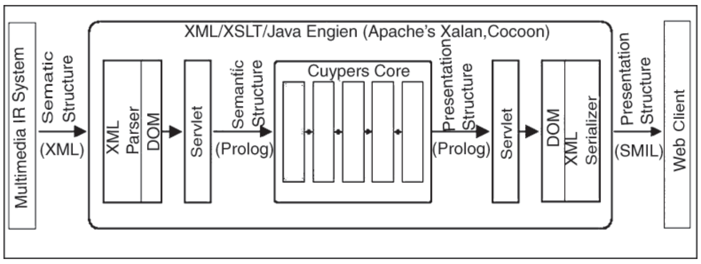


Fig. The Core Cuypers Architecture and its Integration within the Apache HTTP Server.

The server parses XML input, using the XML parser of Apache's Xerces framework. The result is, via the DOM interface, converted by a Cocoon Java servlet to an equivalent Prolog term. This Prolog term is the actual input taken by the core of the presentation engine, which consists of a number of transformations written in ECLiPSe. ECLiPSe allows the transformations to combine, within a single runtime environment, standard Prolog rule-matching and back-tracking with high-level constraint solving techniques.

This allows high level transformation rules to generate alternative layouts using lower-level sets of constraints. Layouts with constraints that prove to be insolvable automatically evaluate to false and cause the system to backtrack, trying alternative layout strategies. In addition, the layout rules can exploit Prolog's unification mechanism as a powerful and extensible selector mechanism, without the need to implement a special purpose selector language such XPath . When the constraints for a given layout can be solved by ECLiPSe, this solution is returned back to the servlet. The servlet converts the result back to XML (in this case SMIL), again using Cocoon's DOM interface.

The use of rhetorical structures as the main technique for describing the input, and on SMIL for describing the final-form output. The core of the Cuypers presentation engine, however, is independent of these formats. Any input that can be transformed to a set of communicative devices can be supported by plugging in a rule set that transforms the input to a set of communicative devices. The same applies to the output format, which can be modified by adapting the lower-level rules that use the (solved) constraints to generate the final form output.

The constraints we currently use for the temporal dimension are based on the temporal relations defined by Allen: *equals*, *before*, *meets*, *overlaps*, *during*, *starts* and *finishes*, with similar relations for the spatial dimensions X and Y. For the stacking order of the media items (the "Z" dimension), we use *above* and *below* constraints. Properties of these qualitative constraints, such as symmetry (A below B <==> B above A) or transitivity (A during B and B during C ==> A during C) are described using the Constraint Handling Rules (CHR,) library of ECLiPSe.

Related Work

Generation of synchronized multimedia presentations from higher level descriptions is not novel in itself. Spatial and temporal constraints for specifying multimedia are used, for example, in the Madeus system. We share our objective to realise cross-platform and cross-media adaptation with preservation of the intended semantics with the ZyX document format. While we generate the entire document flow on the basis of a number of knowledge sources, the ZyX approach is essentially based on augmenting an existing document so that it can be adapted while preserving the main information flow.

Within the AI community, a common reference architecture for model-based multimedia presentations has been developed. This Standard Reference Model for Intelligent Multimedia Presentation Systems (SRM-IMMPS) is based on the synthesis of media content, while we focus on reusing existing content from an annotated multimedia repository. Other relations between SRM-IMMPS and our work are described in.

Our work is also closely related to the work of Elisabeth André, who described the use of AI planning techniques in combination with constraint solvers in her WIP and PPP systems. The main contribution of our approach is that it integrates the several processing steps into a single runtime environment so that the system can freely backtrack across the different levels. This allows high-level presentation decisions to be re-evaluated as a result of constraints that turn out to be insolvable at the lower levels (*e.g.* pixel level). Nevertheless, the individual levels remain conceptually separated, which allows the definition of small, declarative design rules instead of the single hierarchy of planning operators used by André. In Cuypers, semantic relations such as the rhetoric structure are encoded as an explicit level of abstraction, whereas these are used within WIP as implicit design guidelines for the implementation of the generation plan. Additionally, Cuypers uses ECLiPSe as a commonly available, off-the-shelf logic constraint programming (CSP) tool while WIP used a dedicated planner.

Future Work: Towards Third Generation Multimedia

Similar to third generation textual content, third generation multimedia will focus on machine-processable content. Richly annotated multimedia presentations will not only facilitate intelligent Web retrieval and brokering services, but also facilitate reuse of media content in other presentations. In the long term, when there is a sufficient amount of annotated multimedia available, systems such as Cuypers would be able to operate without the multimedia database, and, instead, operate directly on multimedia content found on the Web.

Note that W3C document formats such as SMIL and SVG already anticipate this by allowing documents to contain embedded annotations. In addition, ISO's MPEG-4 [MPEG4] also allows embedded annotations. It remains unclear, however, which annotation languages are the most appropriate, and we are currently investigating various alternatives for the encoding of our metadata. We are investigating not only the use of RDF-based languages such as RDF Schema and DAML+OIL, but also approaches that build directly on top of XML Schema, such as the description schemes developed by the MPEG-7 community.

Adequately annotated multimedia is a key pre-requisite for this multimedia variant of the Semantic Web. Unfortunately, current multimedia authoring tools provide little support for producing annotated multimedia presentations.

Much of the underlying semantics of the overall multimedia presentation and the media fragments it contains remains implicit and is only present in the head of the author. In contrast, in the Cuypers system, it is relatively easy to generate such annotations automatically. Since the entire presentation-generation process in the Cuypers system is based on explicitly encoded knowledge, this knowledge can be preserved and encoded as rich metadata annotations in the final-form presentation.

Note that such metadata annotations can arise from different knowledge sources and describe different abstraction levels. For example, when the system is used to generate richly annotated SMIL, the metadata section of the SMIL document may contain metadata about the individual media items (as retrieved from the underlying media database), the rhetorical structure of the overall presentation, domain-specific knowledge of the application, etc.

It could also generate a report of the design rules and user profiles that were used to justify the chosen end-result (*e.g.* the machine-readable equivalent of “this presentation contains much hi-end video because it is generated for users with a broadband network environment”). This could, for instance, be used by the browsers to help with automatic detection of errors in the settings of the end-user’s profile.

While our future research will focus on generating richly annotated multimedia presentations, we are also looking into extending the Cuypers engine to generate other presentation types, including SVG and VRML. In addition, we are currently working on interfacing the engine with the Mirror multimedia information retrieval system. In particular, we are working on improving the automatic generation of the semantic structures (such as the rhetorical structure used in the example).

This generation process should not only take into account a semantic model of the application domain, but also some form of discourse model to provide guidelines on how to convey subjects from that domain to the user.

The current implementation already uses a declarative encoding of the design, user and platform knowledge. These different types of knowledge are, however, still intertwined. This part of the system needs to be redesigned to be able to manipulate the different types of knowledge through interfaces that are tailored to the different tasks and roles of the users that will need to control them, and to be able to encode the required knowledge in a declarative and reusable way. We expect the Semantic Web to play a key role in achieving these goals.

4

Animation

Animators are frequently given a bit of a brush off from the rest of the entertainment community. Much of what's animated is aimed at children or young adults so can't be serious according to some. But take a close look at many animation classics, even for children, and great animation does reverberate emotionally in the viewer. Look at Dumbo, Pinocchio and other Disney classics. Warner Brothers and other studios also brought engaging animated characters to life. These days Pixar and similar 3D animation studios are accomplishing the same thing.

Animators on visual effects projects must achieve a level of realism beyond what happens in most animated films. One's not better than the other but there are differences. A visual effects animator may have to animate a horse or other animal and make them totally believable as the animal they are supposed to portray. In many cases the animation may be intercut with the footage of the real animal so the match in motion has to be spot on. Visual effects animators may also be called on to animate fantasy creatures or talking, breathing characters.

A true test for a character animator is to animate a simple flour sack drawing or model. Even without a body or face an animator can bring life to the flour sack in such a way to convey happiness, sadness, curiosity and other emotions.

Even though the term may be computer animation an actual artist is the one that does the animation. The computer is just the tool as a pencil was to pre-computer animators. Casting the right animator can be as important as casting the right actor.

Puppets

When I worked at ILM some of the modelers were also puppeteers. As puppeteers they were also in SAG since the actions of a puppet was acting.

So when we did leaping laser printers or other things that required puppeting, that was under SAG agreements. I'm assuming the Muppets and any other key puppets would qualify. In this case it's still considered acting when moving an inanimate object with hands and rods. But there is a direct connection between the actor and the final performance.

Acting and editing

The act of editing the film of a performance can change it's impact.

The director guides the actor's performance on set and selects the most appropriate take of the action. Then in editing specific intercutting is used to both tell the story and to emphasis the character as desired. In the early days of film there was a classic test done where a shot of an actor with a neutral expression was intercut with various scenes. The audience response was the actor was doing a good job of showing happiness, sadness and other emotions based solely on what it was intercut with.

Manipulation

The development of digital effects technology and artists proficient in its use allows actual manipulation of an acting performance. On STAR WARS: THE PHANTOM MENACE there were scenes that George Lucas requested be modified. An actor might have looked up in a take but the shot was run in reverse so the actor looked downward instead since that was deemed better for the cut. In some cases the scene was split and the sync of one side was slipped relative to the other. This was used to shift the timing of a reaction from one actor to another. In other cases eye blinks were added or removed as desired. Some directors have added tear drops or other modifications to a performance.

The point here is that even a live action performance can be modified in post-production in an effort to create a better film experience. What was the actor's truth on set may not be what appears in the final film. And in most cases the actors would probably not be aware of it. In the future we'll likely see even more of this as directors and studios seek to take full advantage of the editing process. And don't think that these types of details are beyond the scope or budget of post-production. Visual effects is already heavily used for things like removing wig netting, making adjustments to makeup (to the tune of over $1 million in some cases) and adding bruises or wounds. (That's the real meaning of the term 'digital makeup'.) A few actor adjustment shots will be a drop in the bucket compared to the other work already being done. It's possible at a future time there will be debate about the implications of these types of modifications and what that means to actors.

The real process

Some actors talk or write about motion capture and 3D animation as 'painting' in the image. Painting? Really? That would make sense if you took a time machine back 100 years before computers and tried to explain it I suppose. But these days it makes as much sense as saying Michelangelo 'doodled' the David sculpture.

Short form: The performance of the actor is recorded and applied in some form to a Computer Graphic (CG) animated character.

Long form: Potentially dozens of people create a very detailed and fully articulated CG model. Imagine making a Madam Tussaunds wax figure but an order of magnitude more difficult. Character/creature designers, concept artists and animators work with the director to develop the look of the character. In some cases a physical model is sculpted out of clay for the director to review. Next it needs to be modeled head to toe, down to every key wrinkle in the computer. This is part modeling and part sculpting. A type of skeleton is built where every movable joint that is needed is included and each joint has a specific range of motion. The irises in the eyes can open and close, the eyes can move, the chest can expand when breathing, nostrils flare and the tongue is configured so that speech looks correct. In many cases muscles are built and configured to change shape. Texture artists paint every surface of the model, not with just one paint process but with multiple paint versions. One paint version shows how shiny different parts of the model are. Another is used to show dirt and still another may be used to show the subtle skin textures and tiny wrinkles.

Fur and hair may need to be added. The angle of the hair, the length of the hair and style of the hair have to be modeled. Another person may be responsible for defining where things are hard (finger nails, shells, etc) and where things are soft and pliable on the model. A shader writer writes specialized programming code to make the skin look real in different types of lighting and that it have specific translucency depending on where it is on the body. Every major facial and phonetic expression needs to be modeled to aid the animator in adjusting and expressing both emotion and voice. All this is done by a team of skilled artists and technicians who use the computer as a tool as a sculptor may use a chisel.

Every item of clothing and every prop the character handles has to be designed and built. The motion of the clothing is setup to be simulated. Does this clothing item behave like silk, cotton or canvas? Dangling earrings and the hair will be programmed so that they move in realistic manner when the character is moving.

All of that just to create the character model.

Animation Process

A good animator does act out the scene at their desk or within their mind visualizes the performance. Unlike a live action actor, the animator has to also observe and analyze every motion and expression change. And not only the position but also the timing of all of those motions need to be noted. The animator is able to visualize the changes in slow motion, forward and backwards. They have to translate that into key frame positions for the animated character model. If choice of the position of the arm or the eyebrow is wrong or if the time of those motions is wrong, the animation won't work. Take after take is required to refine the animation. Any dialog is analyzed as well and the facial expression not only has to convey characters emotion at that 1/24 second granularity but the mouth shape, lips and tongue have to match the correct shape for the phonetic sound created in that moment.

Even actions like setting the feet down and walking take work on the computer. Activities in the real world that happen naturally take effort to do on the computer. The ground in the computer has to be built and matched to the ground of the real location and this is done by a person (match mover). If a foot goes too low then it will go into the ground and if it's too high the foot won't make contact with the ground.

And once the animation is done someone has to light it much like a Director of Photography does. Someone has to render the character and others have to composite (combine) the image with the original image from the set. There's an entire team of craftsmen as big or bigger than the live action crew to make all of this happen.

Performance capture

The combination of powerful computers and digital video cameras made it possible to do computer vision. Images and motion could then be analyzed for scientific and medical purposes. The visual effects industry, as usual, looked to take advantage of these new technologies. Motion capture (MOCAP) became a way to reasonably capture 3D motion data, especially human motion, into the computer. This is useful for recording basic human motion for action shots. Facial capture has been developing which makes it possible to capture not only body motion but the entire performance. When the motion captured includes all aspects of a performance then the term 'performance capture' may be used instead of motion capture.

The typical system uses multiple markers that are strategically placed on the actor. A number of specialized cameras are placed around on a small to medium sized stage that is lit by subdued light. The multiple views are combined by special computer software to yield 3D information on each joint movement. In these cases only the motion was captured after the main photography had already been complete. Developments over the last few years (VAN HELSING, etc.) have allowed the capture of an actor on the stage or outside while being filmed at the same time. Additional simplified motion capture processes have also been developed (PIRATES 2, etc.) and it's now also possible to motion capture a number of actors at the same time. These advancements have allowed motion capture actors more freedom and more interaction with fellow actors. (We still use tennis balls and other references at times. If the CG character is only 4 inches tall or is over 20 feet tall then it can be difficult for a real actor to stand in place and provide the correct interaction.) For interactive performance capture actors are fitted with special suits and act with fellow actors who will remain in the final scenes. This interaction is of course beneficial for both the actors and the director. Another team of visual effects artists then go through and remove the performance actor by literally painting and restoring what would have been behind them. This is a very labour intensive and time consuming process since it involves hand painting frame by frame and creating imagery that isn't in the original. The animated character or creature is then rendered and composited into the scene.

The problems

It might seem that once the motion data is captured it could simply be applied to the CG model and viola - a moving character that exactly matches the performance of the actor down to the smallest detail. But alas, such is not the case by a long shot. If it were then many of the animated films created today could be done using performance capture but they aren't. Live action and animated movies are different art forms and what we're seeing in some cases is a hybrid of the two. There's still plenty of growing pains.

- Even with improvement in the motion capture process there is quite a bit of cleanup required. A simple motion of an actor reaching out may have some frames where the arm leaps up or down a few inches. There might also be random frames where the arm goes behind the actors back or through his body. This takes a small team of people to go through and remove these glitches and clean up the data such that the performance is as pure as possible. In some cases large chunks of data may be missing which requires an animator to fill in with the appropriate motion.
- CG characters seldom match the real actor unless it's a digital double. Frequently the proportions are changed such that the arms or legs maybe longer or shorter. This means that the stride and interaction of what the actor was doing doesn't match what it should. The further away this gets from the real actors body the more difficult it becomes to use the data as it is. A Satyr has totally different leg joints and needless to say a four-legged creature or caterpillar can render much of at least the body motion useless. Captured facial performance may not have much use when the facial structure is very different such as on an insect like creature. All this requires an animator skilled in the understanding of motion, performance and animation to try to retain at least a sense of what the original actors performance was providing. In some cases even performance capture data may become a basic reference or simply inspiration with the brunt of the performance created by the animator.
- Eyes are the window to the soul and a key tool for the actor but actually capturing the eyes, eye blinks and iris

changes has not happened in a meaningful way. That means it's up to an animator to add these types of fine, but critical, details to complete the performance.

- In the editing and visual effects process it may be determined that some adjustments may need to be made for technical reasons. Placement and timing of the character versus what the performance capture actor was doing. In some cases it may be the directors creative call to modify a performance once it has been reviewed with the rendered creature in place.

The end result is it's likely a fair bit of performance capture undergoes some manipulation and work by animators and other artists. In some cases it may be inappropriate to use performance capture simply because the amount of work required is large and the amount of the performance that can be retained is small. Each project has to be evaluated dependent on the creature/character and how cleanly the performance capture can be used.

So you have pure live action acting on one end of the spectrum and pure animation from scratch on the other end. And in-between you have a gray area. One step away from pure animation is to use references. For DRAGONHEART the animators used both stills and clips of Sean Connery to try to incorporate a bit of his personality into the animated performance they were creating. For a project like RANGO they filmed the actors going through the scenes and the animators used this as a reference to create their animation, trying to keep the spirit and emotion of the actors. In some cases they probably followed the actors performance very closely and in other cases they may have ended up deviating quite a bit from the recorded reference. In this film the types of characters (based on a range of animals) would make it impossible to do an exact match of an actor's performance. It's up to the animator to re-interpret and adapt the actors art form into an animated art form, much as a screenwriter may have adapt a novel into a screenplay. The core insight may remain the same but changes have to made to deliver it in a different form.

THE IMPACT OF CGI ON 2D CEL ANIMATION

We proceeded as all artists did before us: with pencil and paper. If anybody wants to be an animator, they should learn to draw the human figure.

- Chuck Jones

If CGI competes head-on with stop-motion, it also competes with 2D cel animation. Largely aimed at the same audience, they are thus competing for the budgets that studios are prepared to spend reaching that audience. There is also a more subtle form of competition that occurs within the animation community itself, where a relatively fixed number of practitioners have to choose which tools they will use to realise their ideas. There is considerable evidence of a growing drift of animators towards CGI.

As well as competing, CGI has also been used with cel animation in a co-operative manner. In the late 1930s, Disney developed the 'multiplane' camera system, an elaborate animation stand that allowed several separated cel layers (foreground characters and background sets), to be moved independently frame by frame, giving a powerful illusion of three-dimensional space. An updated version of this technique uses CGI to replace the background layers.

Films produced using this approach are sometimes called 2D/ 3D hybrids. In such films the foreground characters are handled in a conventional 2D manner (though often using computers to assist the animators to do tweening) but background scenery -

buildings and trees for example as well as features such as crowds - are modelled using 3D CGI. This is becoming an increasingly popular way of producing 'traditional style' cartoons - Disney used CGI for backgrounds and crowd scenes on Mulan (1998) for example. Several of the forthcoming features mentioned in Appendix 4 are of this type.

One advantage of this approach (which it shares with 3D CGI) is that it makes it simple to adjust the position of the camera and even move it during a shot. While the 'multiplane' system allowed the camera to perform tracking and zooming shots, it could not cope with a true pan because rotating a constant background image introduces unacceptable perspective distortion. CGI offers much more freedom because the background is redrawn in the proper perspective for each frame.

Evidence for the impact of CGI on 2D cel animation is not hard to find. The success of the four features Pixar has so far produced (Toy Story (1995), A Bug's Life (1998), Toy Story 2 (1999), and Monster's Inc. (2001)) has not been matched by Disney's own traditional 2D offerings. The following table shows the budgets and world-wide gross box-office figures for the four Pixar features.

Title	Date	Budget	World-wide Box Office
Toy Story	1995	$30 million	$356 million
A Bug's Life	1998	$30 million	$358 million
Toy Story 2	1999	$90 million	$486 million
Monster's Inc.	2001	$115 million	$523 million

Pixar 3D CGI Animated Features since 1995

The second table shows the budgets and world-wide gross box-office figures for the seven Disney animated feature released since Toy Storyappeared in 1995.

Title	Date	Budget	World-wide Box Office
The Hunchback of Notre Dame	1996	$70 million	$303 million
Hercules	1997	$70 million	$250 million
Mulan	1998	$70 million	$303 million

Tarzan	1999	$150 million	$435 million
The Emperor's New Groove	2000	$80 million	$160 million
Atlantis: The Lost Empire	2001	$90 million	$139 million
Lilo and Stitch	2002	$80 million	$190 million

Disney 2D Animated Features since 1995

It is dangerous to read too much into these tables, but in contrast to the Pixar figures, there has been a clear reduction in gross box office takings for Disney's traditional offerings over the last few years. The apparent success of Tarzan is countered by its very high budget, which in turn was due to the very large number of artists (over 570 at the peak) who worked on the film. It was from this point that Disney started making dramatic cuts in its overheads.

In March 2002 the Wall Street Journal reported that the company would lay off an additional 250 animators over the course of the year. When Lilo and Stitch was released in June 2002, Thomas Schumacher, president of Walt Disney Animation, said that Tarzan would have been a considerably greater financial success if it been made with the lower salaries and cost-control measures now in place.

Tarzan generated an internal rate of return of 14 per cent on Disney's investment but using the new processes the return would have been 35 per cent.

Dreamworks SKG, which commissioned and distributed Antz (1998) and Shrek (2001), the two very successful CGI animated features produced by PDI, has also made traditional animated features itself in the last few years. As with Disney, these traditional features have been failing to match the box office success of the CGI releases, as the following tables show.

Title	Date	Budget	World-wide Box Office
Antz	1998	$60 million	$152 million
Shrek	2001	$50 million	$482 million

Dreamworks SKG (PDI) CGI Animated Features

Title	Date	Budget	World-wide Box Office
The Prince of Egypt	1998	$60 million	$221 million
The Road to Eldorado	2000	$95 million	$75 million
Spirit: Stallion of the Cimarron	2002	$48 million	$97 million

Dreamworks SKG 2D Animated Features

Again it is dangerous to draw too many inferences, but there is a pattern here too. Note that both Road to Eldorado and Spirit are 2D/3D hybrids, using CGI for backgrounds and crowd scenes.

The Impact of CGI on Stop-Motion Animation

Model animation, by its very process, has a slight unpredictability and spontaneous feel to it - even the animators cannot exactly predict where the puppet will go - and this does give it a unique edge.

- Barry Purves

I don't think after Jurassic Park that we can, or should, ever accept a model-animated dinosaur again.

- Barry Purves

It should be obvious that although the success of 3D CGI cartoons is likely to affect all conventional forms of animation, it is with stop-motion that it competes most directly, and hence will impact most. After all, it is already possible, using CGI, to reproduce the vast majority of conventional stop-motion sequences. Further, Moore's Law continues to make CGI cheaper, while stop-motion costs are likely to increase.

Things are rarely simple however. As was mentioned in the introduction, the tools an artist or craftsman uses can have a profound effect on their way of working. How do the undoubted differences between the techniques affect the end results? And do they affect the imagination and ideas of the people involved?

There is quite a lot of anecdotal evidence that stop-motion exerts particularly strong effects, both psychological and physical, on its practitioners. Adherents feel that the limitations and idiosyncrasies of the technique lend a very particular character to the end-result. Following a panel discussion of stop-motion animators, Stephen Arthur made these remarks:

Barry Purves inspired the audience with his impassioned descriptions of the sensual art of "directing" his elaborate puppets. He let us handle these expensive puppets: the original alien from Mars Attacks, the statue-like figure of Achilles, and his "14-inch Willy" (Shakespeare) from Next. It was an astonishing feeling, a very direct connection with the "actors" in puppet animation. Puppets are usually held intimately by the animator's hand for every increment — a strong contrast to the separation from the animated flow experienced with most types of animation. I could suddenly understand what entices so many animators into puppet animation.

Purves himself goes even further:

Because of the one-to-one relationship between an animator and puppet in the actual process of animation, a lot of the character and the passion of the animator himself goes straight into the puppet and its performance. It is the most purely personal and honest form of animation (often revealing surprising aspects of the animator's personality), and acting is the most undiluted, whereas with drawn and computer animation, it is, to some extent, animation by committee. There are, of necessity, so many more people involved, and something really does get lost on the way.

There is a close relationship between acting and animation, indeed it has been said that animators are often frustrated actors who are too frightened to act in front of the camera, so they act with a pencil or a puppet instead.

Because successful animation requires a deep understanding of how and why animals move, animators often take acting classes to develop their understanding of the relationship between mind and body. In stop-motion, the direct physical manipulation of the puppet makes animation much more of a performance. As Peter Lord says:

[Plasticine animation] is a technique that's perfect for improvisation.... You have your camera, you have your puppet, and your animator knows the intention of the shot, but there are so many ways you can carry it out. So it's like any other performance. You may know you've got to keep the shot within three seconds, and it may take you a whole day to do, say, a second and a half, but you have so many opportunities throughout that day to change your mind on how the shot goes. And ideas come to you, because you've got all that time. With drawn animation, it's like you're creating a more sophisticated flip-book, so you've got to look through your drawings and keep going back and adjusting. And with computers, you can keep changing the animation and layering it..... Here, you've got a puppet and you're just working forward, and if it doesn't quite deliver, you start again from the beginning. You really live in the moment, like an actor would on a stage. Each day is a singular performance.

Supporters of stop-motion argue that CGI technology, where the puppet is a virtual figure, seen on a computer screen and manipulated though a mouse or keyboard, lacks this direct, tactile approach, and hence cannot duplicate the particular qualities of stop-motion. This need not be the case, as the following case study shows.

The Tyrannosaur in Jurassic Park (Spielberg, 1993) was originally intended to be filmed using a combination of a large 'animatronic' robot (built by Stan Winston) and conventional stop-motion models (animated by Phil Tippet). CGI was to be used purely for the stampede scene, which involved too many animals to be filmed using models. The early attempts to animate a Tyrannosaur using CGI were so impressive that Spielberg abandoned the idea of using stop-motion, although he did retain Tippet as a 'dinosaur supervisor'. Tippet organised classes in mime and field trips to zoos for the computer animators, but he also invented an ingenious computer input device, the Dinosaur Input Device or DID. He equipped the joints of a conventional ball-and-socket stop-motion armature with encoders that allowed any movement of the armature to be communicated to a computer. In this way a CGI model of a dinosaur could be manipulated using the techniques of traditional stop-motion.

The DID was short-lived - by the time work on the film was completed the computer animators had switched back to using mouse and keyboard - but it does serve to demonstrate that CGI need not be an entirely 'virtual' activity. CGI Animators can physically touch and feel their creations through such exotic haptic devices.

In any case, even when using a mouse and keyboard, aspects of the animator's personality still get transferred to the screen character, as Andy Jones, animation supervisor on Final Fantasy: The Spirits Within (Sakaguchi/Sakakibira, 2001), explains:

Roy Sato, for example, was the lead animator on Aki [Dr. Aki Ross, one of the leading characters in the film] and you see a lot of Roy in that character. The timing of Aki's blinks are the same as his....

The most obvious demonstration of the serious impact that CGI can have on stop-motion animation lies in the contrast between

Nightmare and Tim Burton's subsequent film, Mars Attacks! (1996). Though Nightmare was a stop-motion triumph, its box-office performance was not outstanding - it cost around $23 million and made just $57 million. Here are some statistics that may indicate how the money was spent:

13 principle animators

100 model makers and set builders

230 sets - often 20 or 30 feet square- on 19 stages

60 characters (more than 200 puppets)

400 heads just for Jack Skellington

1 minute of film completed each week

400 frames in a typical shot (which took 10 days to shoot)

Burton originally intended the Martians inMars Attacks! to be animated by Henry Selick using the same stop-motion techniques they had employed on The Nightmare Before Christmas. Selick's company, Skellington Productions, was however still busy on James and the Giant Peach (1996) so Burton turned to two English model makers, Ian Mackinnon and Peter Saunders, whose credits included the creation of puppets for the Oscar-winning stop-motion short The Sandman (Berry, 1991).

Mackinnon and Saunders organized a large team of sculptors in Los Angeles and the U.K., building hundreds of identical 15-inch Martian puppets. Mackinnon was soon joined by puppet animator Barry Purves, creator of the award-winning short films Next and Achilles. With Purves acting as animation director, elaborate sets were constructed and filming began. After months of work designing Martian gestures and ways of moving, Warner decided that blending stop-motion animation convincingly with live-action was too challenging a task to be dealt with in the year left before the film's scheduled release. The model animation team was dispensed with, to be replaced by CGI animators from Industrial Light and Magic, although not all of the model work was discarded. The movements and gestures developed by Purves' team were adapted to the computer characters, the puppets were digitally scanned and rendered into computer models, and scaled up to build full-scale Martians for several of the film's live-action scenes.

Mars Attacks! was an expensive film to make, and not just because of the money spent on discarded stop-motion. Unlike Nightmare, which had no expensive stars, Mars Attacks! had a large cast which included major stars such as Jack Nicholson, Pierce Brosnan and Glenn Close. It also featured several large and expensive set pieces. Eventually costing over $70 million, it failed to recoup its costs, with a US box office gross of less than $40 million. As far as the public, critics and industry observers were concerned however, the failure of the film was not due to the use of CGI, but in spite of it - most felt the special effects were outstanding.

As has been mentioned, Henry Selick was working on James and the Giant Peach (1966) while Mars Attacks was in production. Selick's plan was to shoot the central character as a live action boy and have him interact with stop-motion creations through the entire story. When that proved too expensive he explored making every element stop-motion. Disney felt the cost was too high, so a compromise was reached. Of the result Selick has said:

The compromise was hard on a lot of audiences. I believe it would have worked better either way, all stop-motion or as James all live throughout.

Before James was released, it was widely felt at Disney that it would do better than Nightmare. As can be seen from the figures in Appendix 2, it actually did much worse. The abandoning of stop-motion on Mars Attacks! and the unexpected box-office failure of James convinced many in the industry that CGI was the way of the future, although Selick was still optimistic. However the collapse of a three-picture contract with Miramax in 1997 led him to close his studio in San Francisco and work on smaller, personal projects. The dismal box office performance of his latest film, Monkeybone (2000), which, like James, mixed live action and stop-motion, has discouraged most American studios from becoming involved with stop-action features.

The exception is Dreamworks SKG, which financed the very successful Chicken Run (Lord/Park, 2000) as part of a 250 million dollar, five picture contract with Aardman Animation. It is worth asking what is the secret of Aardman's success and whether they will be able to continue making feature-length 3D animated films using stop-motion when everyone else has switched to CGI.

Part of the answer surely lies in Aardman's success with its Wallace and Gromit series. These three half-hour films have been extremely popular on television all over the world and, along with Creature Comforts, account for Nick Park's three Oscars. While the income from these short films has been minute in comparison with that from feature film successes like Toy Story or Chicken Run, they have created a very large, world-wide audience that is familiar with, and attracted to, the Aardman style. As Peter Lord says:

We've kind of developed this style with the eyes right together in the middle and a big wide mouth, just because that plays well. People find that incredibly funny for some reason. Just the still image makes them laugh, so we have to go along with that.

As well as strong stories, the emphasis on humour is something that Aardman shares with Pixar. Both avoid the excesses of sentimentality characteristic of much of Disney (as well as the saccharine songs) and the dark expressionism that characterises much of Selick's work. Both share a delight in visual and verbal puns, in the spoofing of film genres and in rich visual detail. Indeed, there is a strong affinity between the 'look and feel' of Aardman's plasticine and that of Pixar's 'virtual plastic' (it is not just the toys in Toy Story that appear to be made from plastic, the insects in A Bug's Life do too). It is not surprising that both companies have a profitable sideline in merchandise - toys, T-shirts and mousepads, as well as books, videos and DVDs - based on key characters such as Wallace, Gromit, Woody and Buzz.

Successful as Chicken Run was, something may have been lost in the transition from shorts to features. The large number of characters involved meant that puppet making became more of a production line, with the use of moulded plastic instead of plasticine for some chicken body parts. With so many people working on the film (upwards of 200), keeping the animation consistent inevitably resulted in a lessening of the stamp of individuality that characterised the earlier, shorter pieces. As Nick Park recalled:

I remember starting A Close Shave. There were more animators than ever working on that one, and they were all put through Wallace and Gromit lessons, where they learned how to move the brows and the eyes and the mouths. We did that even more on Chicken Run. Every Monday night, we had workshops so everyone would handle the characters in the same way.

Aardman's next feature for Dreamworks will be The Great Vegetable Plot, featuring Wallace and Gromit, and is due for release in the autumn of 2004. The company is also busy working on a variety of shorts, including a new series of Creature Comforts for ITV and a series of 1-minute Wallace and Gromit episodes to be released via the Internet.

Not everything at Aardman has been going as planned. The second Dreamworks feature was to have been The Tortoise and the Hare, based on Aesop's fable.

Originally scheduled for release this summer (2002), and then in 2003, it now looks unlikely to appear before 2005. In July 2001 Aardman announced that it had laid off 90 of the 170-strong crew because of script problems on the film. The Hollywood Reporter announced on June 11, 2002 that the film had 'failed to cross the finish line after script troubles', implying the project had been cancelled. Then on July 24, 2002, a spokesman for Aardman Animations revealed that work was continuing on the script for the film, which is now expected to have CGI animation augmenting its stop-motion work.

Despite their commitment to stop-motion, Aardman have been experimenting with CGI, though Peter Lord has said that trying to copy clay animation exactly using CGI would be a 'very sterile exercise because it is just copying' - what interests him is 'devising a new language' for CGI animation. As he says:

Well, there is something about working with the materials. There is a fundamental difference between working with your hands and your arms and your fingertips, and working on the keyboard..... You grab the puppet with two hands, and you feel the whole thing move, you feel the twist of the chest away from the hips, the roll of the shoulders....

If stop-motion animation has had problems, disappointments and failures, so has CGI animation. As well as the disappointment of Mars Attacks!(which had a foot in both camps), there has been one CGI failure of truly epic proportions. Final Fantasy: The Spirits Within (Sakaguchi/Sakakibira, 2001) had a budget of $137 million dollars, but has so far recouped just $32 million dollars.

Final Fantasy is perhaps the most ambitious CGI feature to date. The film, which took four years to make, put synthetic human actors into roles that could easily have been played by real humans and placed them in completely synthetic sets. The film, based on a series of hugely popular, interactive, role-playing computer games, was produced by Square, the company that produces the games, and co-directed by Sakaguchi, the game's originator. With world-wide sales of the nine-part game series totalling more than 26 million units, Square must have thought it had a ready-made potential audience of game-players familiar with the fantasy themes, comfortable with computer graphics characters, and eager to see the next installment.

The CGI animation has been rightly regarded as a technical triumph, so why was Final Fantasy such a commercial failure? The answer, at least as far as western audiences and critics were concerned, was the weakness of the story and the lack of pace in the way it was told. Lacking a compelling plot, the glamour that human stars can bring to a film, and without much humour, the film had little but its special effects to hold the attention of the audience. As a result of its failure at the box office, Square announced in February 2002 that it was closing the studio in Hawaii that had created the film.

Although Disney has relied on its partner, Pixar, to produce CGI cartoon features that it then distributes, it has made one attempt at a (partial) CGI feature itself, Dinosaur (2000). Although the dinosaurs in Dinosaur were created using CGI, the scenery was live action, with backgrounds shot all around the world. The film made $350 million at the box-office, but it cost $200 million to make - at one time nearly 900 people were working on it. Its poor financial performance caused Disney to close the CGI unit it had created to produce the film, though it has since re-opened it on a much smaller scale. The accepted explanation for the failure of Dinosaur is the familiar one - a poor plot - which recycles ideas from The Lion King and Tarzan.

5

Acting and Animation

There is a common saying that, "An animator is an actor who uses a pencil and paper rather than their body to give a performance." Animators, by-and-large are frustrated actors who can't really perform physically in front of an audience for various reasons. Many of us tend to think of ourselves as "geeks" (well, that's how I feel anyway). Our physical bodies are limited in what they can do or how they look. An actor such as Robert DeNero can portray boxing champion Rocky Marciano in one film, then a Taxi driver, a young Sicilian mob boss, or a father-in-law in others. He has versatility and great acting skills as well as a very good manager. The same can be said for many other well known actors and actresses.

Animators however, have far more versatility than could ever be imagined by any live action actor simply because the only limitation is their imagination. An animator can become anything they want... anything. The only real limitation is their ability to a) draw, and b) animate. The two go hand-in-hand but are not completely bound. A person with limited drawing ability will not be able to portray a "realistic looking" human but if they know how to animate really well, they could give their drawings some truly exceptional character acting.

On the flip side, someone who knows how to draw incredibly well, (let's say like Michelangelo) but probably like Michelangelo, can't animate worth beans, you'd get some pretty ugly looking movements and hence bad acting. So what is it about acting that makes animation look so good?

Let's go back in time just for a bit to the mid 1800's. There was this guy in France named Francois Delsarte. He was born in 1811 and died in 1871. He taught acting and singing and ended up developing a theory on acting that relied on dramatic posing and physical signals. Rather than have the actor just stand in the middle of the stage without moving and deliver his lines such as, "My pain is too great for me to bear!", Francois thought it would look great if the actor threw his arm across his forehead while thrusting the other hand, claw-like, forward in front of him and then collapse back against a chair or something. Oh yeah!

In 1885, 14 years after he had passed away, (dramatically, I assume) his theories were published as the "Delsarte System of Expression". Many schools of acting in America adapted the system into their training during the late 19th century. In some early films from 1900 - 1925 you can still see some actors using his system (they're quite funny to watch).

In 1897, Konstantin Stanislavski, a Russian actor, set up his acting workshops at the Moscow Theatre. Rather than just having his actors pose to show their emotions he developed a system called "Method Acting" whereby the actor actually felt something because they had “become the character”. Stanislavski wrote several books on acting. The three which you will probably find most useful are: “An Actor Prepares”, “Building A Character”, and “Creating A Role”. They are excellent books to study. However, as Stanislavski himself said, “You must not duplicate the Moscow Art Theatre (method acting). You must create something of your own. If you try to duplicate, that means that you merely follow tradition. You are not going forward.”

He also said something which I feel applies equally as well to us as animators, “Artists must learn to think and feel for themselves and find new forms. They must never be content with what someone else has done.” “If something excites you, use it, apply it to yourselves, but adapt it. Do not try to copy it. Let it make you think further.”

This is basically the way I want you to view this book. There will be some analysis of existing scenes from animated films, examples of the assignments I will provide to you and suggestions of alternate ways to approach the given scenes. Follow Stanislavski’s advice: “If something excites you, use it, apply it to yourselves, but adapt it. Do not try to copy it. Let it make you think further.”

So, back to our main topic: Acting and Animation.

What is “acting”? By definition, acting or “to act” is a verb: to move to action: ACTUATE, ANIMATE, (interesting) PERFORM, EXECUTE, to represent an incident or an emotion by action, to perform as an actor, to play the part of a character, assume the character of, to behave in a certain way as to convey an emotion, characterization, or certain actions.

So from this we know that acting is the physical performance which conveys:

- A character
- An incident
- An emotion
- Certain actions
- Certain behaviour

In his book, "Acting For Animators", Ed Hooks lists "Seven Essential Acting Concepts" as:

- Thinking leads to movement and emotion.
- Acting is reacting. Acting is doing.
- Your character needs to have an objective.
- Your character should play an action until something happens to make them play a different action.
- All action begins with movement.
- Empathy is the magic key. Audiences empathize with emotion.
- A scene is a negotiation.

Frank and Ollie felt there were three very special problems in the field of acting for animation which could not be ignored. These were found on page 502 of "Illusion of Life".

- The animator must know what the character should do in a particular circumstance.
- They must be skilled enough as a craftsman to capture in drawings what they know in their head.
- They must be able to retain the fleeting delicate thought of the moment over the several days it may take to animate the scene.

Again, in the book "Illusion of Life", on page 137, Frank and Ollie have a list of 12 components that are found in good animation:

- Inner feelings and emotions

- Acting with clear and definite action
- Character and personality
- Thought process through expression changes
- Ability to analyze
- Clear staging
- Good composition
- Timing
- Solidity in drawing
- Power in drawing
- Strength in movement
- Imagination

Trying to find the moment when the audience connects with the character on the screen. They are right there with the character, they understand them and are concerned about what happens to them. It is this moment when the character reaches out and touches the audience.

Getting the audience involved requires an understanding of the character and using feelings that are familiar with everyone.

This doesn't necessarily mean using sympathetic emotions such as happiness or sadness. You can also create feelings of anger, or fear, shock or revulsion. These are the six basic emotions.

In each of the lists above, the common thread is EMOTION. It's all about how you feel. How you feel about the character, how you feel about their circumstance, how you feel about the resolution. If you don't "FEEL" anything, what's the point?

Think about the last time you went to a movie. What kind of movie was it? Was it a comedy? Action/adventure? Horror, suspense, romance?

If it was a comedy, did you laugh at any point in the film? If it was action/adventure did you feel a certain thrill during a chase sequence? Was your heart beating faster? Were you excited? At the horror or suspense movie, were you scared or on edge? During the romance movie, did you feel the heartache or happiness of the united couple at the end?

If you answered "no" to any of the above questions, the movie was a failure and you probably should have asked for your money back. Our choice of movie genre is based on our emotional need at that point in time. You go to the film to feel that emotion, if you don't you were ripped off.

The only way you can feel an emotion is if you empathize with the character in their situation. To empathize, the actor needs to convey the proper actions that relay the appropriate emotion. This is where good acting comes into play. However, do not confuse action with acting. This goes back to our pal, Francois Delsarte back on page 5, remember him? You don't need to overly dramatize the action to make the emotion read. The character must be true to who they are.

Acting

Acting is really all about thought. If you have a character that doesn't think about what they are doing, you really have a puppet. Puppets are inanimate objects that only do what the person who is holding them, or controlling them, wants them to do.

I remember buying a stuffed doll which was also a hand puppet for my daughter when she was 2 years old. She saw it in a bin and fell in love with it. She named him "Charlie Dog" on the spot. When I pulled him out of the bin I noticed that it was also a puppet, so I stuck my hand in and started talking to Jenna in a "Goofy" voice. She didn't really like the idea that he could move and talk on his own so she grabbed him off my hand and hugged him. The moment he came off my hand he became inanimate and lifeless in her arms. The only time he became a “character” was when I did the puppet thing. I had to do all the thinking for him.

After a while, Charlie Dog became the bedtime routine for reading books and Jenna really enjoyed it.

Your animation needs to be treated in the exact same way. Without your concious thought flowing through the drawings and out of that character, the animation will appear lifeless. Many animators neglect this “thought process time” for the character on the screen.

One of the assignments I give my first year students is called “The Phone Call” In this assignment they are to have a character who receives a phone call from another character. The character receiving the phone call is to pick up the phone and answer it. The character on the other end is to say something which causes the first character to change their emotion.

Invariably, the student will animate the scene where the character picks up the phone with one visible emotion, then within half a second (literally) the character changes their emotion. Twelve frames is not enough time for the first character to have registered who the other character is, get the message the other person is conveying, think about it and then change their emotion. The start of any phone call usually goes like this:

Ring, ring.

Character A picks up phone and holds it to their ear.

Character A: “Hello?”

Character B: “Hi, is this characterA?”

Character A: “Yes it is.”

Character B: “Hi Character A, this is Character B. How are you?”

Character A: "Oh, hi Character B, I'm fine, how are you?"

This whole conversation takes about 10 seconds total. Of course the second and third lines could be eliminated but even then this conversation would still take about 6 or 7 seconds.

Character A's emotional change would take place on the last line after they recognize who it is on the other end of the phone. This would be about 5 seconds into the conversation.

5 seconds x 24 frames = 120 frames or 60 drawings (on two's) for the Character A to think about what Character B is saying and how they will repond to it. It is this "think time" that really allows your character to come alive to the audience.

This doesn't mean that your character needs to be doing some sort of action during those 120 frames. It could be a subtle as a moving hold with an eye blink. The idea is to make the character look as though they actually are thinking about something.

Don't Confuse "Action" With "Acting"

Your character needs to have a purpose for moving. Action, just for the sake of action is not a good thing. Of course on the flip side, you can't act something out without some form of action. Try playing charades without moving at all. What you're really trying to do is illustrate an idea or thought with the attitude or actions of your character. Every action your character makes must have a purpose or reason. Any type of movement on screen will draw the audience's attention because they think something is going to happen and they follow the action. If it's distracting the audience may miss the focal point of the scene or become confused. You don't want the character to act like a magician who gestures with one hand while producing something in their other hand, seemingly out of thin air... unless of course, the character you're animating actually is a magician.

Here are 12 questions the animator must ask themselves before animating a scene:

- Is the character doing what the director wants in the sequence?
- Is the character doing only one thing at a time?
- Is the character putting over the story point in the scene you are doing?
- Is the character acting as if there is something going on in his mind?
- Does the character appear to be doing something on his own?
- Can the audience tell what the character is thinking?
- How does what the character is doing effect what the audience is thinking?
- Does the character have appeal?
- Is it passionate? Is passion going into the drawing and coming out of the character?
- Is it the simplest way to do it?
- Have you made small story sketches of one important character to be sure everything is working before you make a lot of drawings?
- Would any one else besides your mother like what you have done?

Animations

Clutter actors have a variety of properties (position, size, rotation in 3D space, scale, opacity) which govern their visual appearance in the UI. They may also have constraints on how they are aligned and/or positioned relative to each other.

The Clutter animation API provides a means of changing properties and constraints as a function of time: moving, scaling, rotating, changing opacity and colour, modifying postional constraints, etc.

Clutter also makes it possible to animate non-visual properties if desired.

High level overview

Here are the main concepts behind animation in Clutter:

- An animation changes one or more properties of one or more actors over time: their rotation in a particular dimension (x, y, z), scale, size, opacity etc.
- An animation has an associated timeline. Think of this as analogous to the "thing" you're controlling when you watch a video on the internet: it's what you control with the play/pause button and what is measured by the bar showing how far through the video you are. As with the controls on a video player, you can play/pause/skip a Clutter timeline; you can also rewind it, loop it, and play it backwards.

If a timeline is reversed, the progress along the timeline is still measured the same way as it is in the forward direction: so if you start from the end of the timeline and run it backwards for 75 per cent of its length, the progress is reported as 0.25 (*i.e.* 25 per cent of the way from the start of the timeline).

- The duration of a timeline (*e.g.* 500 milliseconds, 1 second, 10 seconds) specifies how long its animation will last. The timeline can be inspected to find out how much of it has elapsed, either as a value in milliseconds or as a fraction (between 0 and 1) of the total length of the timeline.

- An animation is divided into frames. The number of frames which make up the animation isn't constant: it depends on various factors, like how powerful your machine is, the state of the drivers for your hardware, and the load on he system. So you won't always get the same number of frames in an animation of a particular duration.
- The change to a property in an animation occurs over the course of the timeline: the start value of the property heads towards some target value. When it reaches the end of the timeline, the property should have reached the target value.
- Exactly how the property changes over the course of the timeline is governed by an alpha.

Alphas

An alpha is generated for each frame of the animation. The alpha varies between -1.0 and 2.0, and changes during the course of the animation's timeline; ideally, the value should start at 0.0 and reach 1.0 by the end of the timeline.

The alpha for any given frame of the animation is determined by an alpha function.

Usually, the alpha function will return a value based on progress along the timeline. However, the alpha function doesn't have to respect or pay attention to the timeline: it can be entirely random if desired.

To work out the value of a property at a given frame somewhere along the timeline for a given alpha:

- Determine the difference between the start value and the target end value for the property.
- Multiply the difference by the alpha for the current frame.
- Add the result to the start value.

The shape of the plot of the alpha function over time is called its easing mode. Clutter provides various modes ranging from CLUTTER_LINEAR (the alpha value is equal to progress along the timeline), to modes based on various polynomial and exponential functions, to modes providing elastic and bounce shapes.

Most of the time, you can use the built-in Clutter easing modes to get the kind of animation effect you want. However, in some cases you may want to provide your own alpha function. Here's an example (based on the quintic ease in mode from clutter-alpha.c):

```
    static gdouble
    _alpha_ease_in_sextic (ClutterAlpha *alpha,
                           gpointer dummy G_GNUC_UNUSED)
    {
    ClutterTimeline  *timeline  =  clutter_alpha_get_timeline
(alpha);
    gdouble p = clutter_timeline_get_progress (timeline);
    return p * p * p * p * p * p;
    }
```

An alpha function just has to have a specified method signature and return a gdouble value when called. As stated above, you'd typically base the return value on the timeline progress; the function above shows how you get the timeline associated with the alpha, so you can apply the alpha function to it.

Clutter's animation API

All of the animation approaches in Clutter use the same basic underpinnings (as explained above), but the API provides varying levels of abstraction and/or ease of use on top of those underpinnings.

- Implicit animations (created using clutter_actor_animate() and related functions) are useful where you want to apply a simple or one-off animation to an actor. They enable you to animate one or more properties using a single easing mode; however, you only specify the target values for the properties you're animating, not the start values.

- ClutterAnimator provides support for declarative animations (defined using ClutterScript). You can animate multiple actors with this approach, and have more control over the easing modes used during an animation: while implicit animations only allow a single easing mode for all properties, ClutterAnimator supports multiple easing modes for each property; key frames are used to indicate where in the animation each easing mode should be applied.
- ClutterState enables you to describe states: property values across one or more actors, plus the easing modes used to transition to those values. It can also be combined with ClutterAnimator for finer grained definition of transitions if desired.

States are particularly useful if you need actors to animate between a known set of positions/sizes/opacities etc. during their lifecycles (*e.g.* animating a list of items in a menu, or for animations in a picture viewer where you click on thumbnails to display a full view of a photograph).

basic principles of animation

Disney's **Twelve Basic Principles of Animation** were introduced by the Disney animators Ollie Johnston and Frank Thomas in their 1981 book The Illusion of Life: Disney Animation. Johnston and Thomas in turn based their book on the work of the leading Disney animators from the 1930s onwards, and their effort to produce more realistic animations. The main purpose of the principles was to produce an illusion of characters adhering to the basic laws of physics, but they also dealt with more abstract issues, such as emotional timing and character appeal.

The book and some of its principles have been adopted by some traditional studios, and have been referred to by some as the "Bible of animation." In 1999 this book was voted number one of the "best animation books of all time" in an online poll. Though originally intended to apply to traditional, hand-drawn animation, the principles still have great relevance for today's more prevalent computer animation.

The 12 Principles of Animation

Squash and Stretch

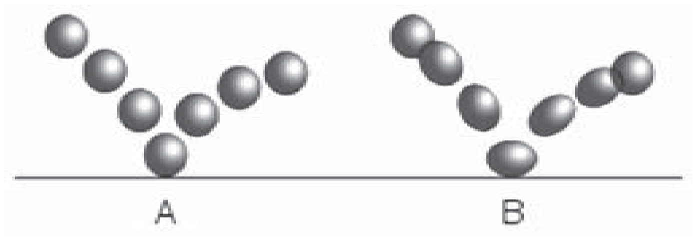


Illustration of the "squash and stretch"-principle:

Example **A** shows a ball bouncing with a rigid, non-dynamic movement. In example **B** the ball is "squashed" at impact, and "stretched" during fall and rebound. The movement also accelerates during the fall, and slows down towards the apex.

The most important principle is "squash and stretch", the purpose of which is to give a sense of weight and flexibility to drawn objects. It can be applied to simple objects, like a bouncing ball, or more complex constructions, like the musculature of a human face. Taken to an extreme point, a figure stretched or squashed to an exaggerated degree can have a comical effect. In realistic animation, however, the most important aspect of this principle is the fact that an object's volume does not change when squashed or stretched. If the length of a ball is stretched vertically, its width (in three dimensions, also its depth) needs to contract cor-respondingly horizontally.

Fig. Animated sequence of a race horse galloping. Photos taken by Eadweard Muybridge. The horse's body demonstrates squash and stretch in natural musculature.

Anticipation

Anticipation is used to prepare the audience for an action, and to make the action appear more realistic. A dancer jumping off the floor has to bend his knees first; a golfer making a swing has to swing the club back first. The technique can also be used for less physical actions, such as a character looking off-screen to anticipate someone's arrival, or attention focusing on an object that a character is about to pick up.

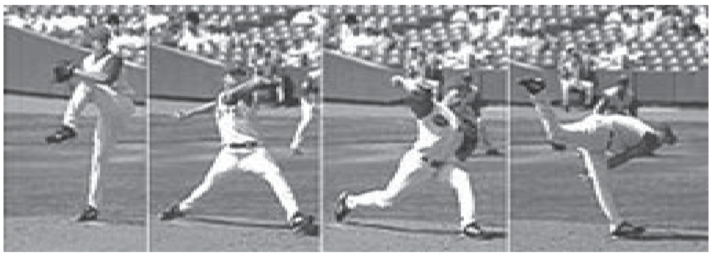


Fig. Anticipation: A baseball player making a pitch prepares for the action by moving his arm back.

Staging

This principle is akin to staging in theatre, as it is known in theatre and film. Its purpose is to direct the audience's attention, and make it clear what is of greatest importance in a scene; Johnston and Thomas defined it as "the presentation of any idea so that it is completely and unmistakably clear", whether that idea is an action, a personality, an expression, or a mood. This can be done by various means, such as the placement of a character in the frame, the use of light and shadow, or the angle and position of the camera. The essence of this principle is keeping focus on what is relevant, and avoiding unnecessary detail.

Straight Ahead Action and Pose to Pose

These are two different approaches to the actual drawing process. "Straight ahead action" means drawing out a scene frame by frame from beginning to end, while "pose to pose" involves starting with drawing a few key frames, and then filling in the intervals later. "Straight ahead action" creates a more fluid, dynamic illusion of movement, and is better for producing realistic action sequences.

On the other hand, it is hard to maintain proportions, and to create exact, convincing poses along the way. "Pose to pose" works better for dramatic or emotional scenes, where composition and relation to the surroundings are of greater importance. A combination of the two techniques is often used.

Computer animation removes the problems of proportion related to "straight ahead action" drawing; however, "pose to pose" is still used for computer animation, because of the advantages it brings in composition. The use of computers facilitates this method, and can fill in the missing sequences in between poses automatically. It is, however, still important to oversee this process and apply the other principles discussed.

Follow Through and Overlapping Action

Follow through and overlapping action is a general heading for two closely related techniques which help to render movement more realistically, and help to give the impression that characters follow the laws of physics, including the principle of inertia. "Follow through" means that loosely tied parts of a body should continue moving after the character has stopped and the parts should keep moving beyond the point where the character stopped to be "pulled back" only subsequently towards the center of mass and/or exhibiting various degrees of oscillation damping. "Overlapping action" is the tendency for parts of the body to move at different rates (an arm will move on different timing of the head and so on).

A third, related technique is "drag", where a character starts to move and parts of him take a few frames to catch up. These parts can be inanimate objects like clothing or the antenna on a car, or parts of the body, such as arms or hair. On the human body, the torso is the core, with arms, legs, head and hair appendices that normally follow the torso's movement. Body parts with much tissue, such as large stomachs and breasts, or the loose skin on a dog, are more prone to independent movement than bonier body parts. Again, exaggerated use of the technique can produce a comical effect, while more realistic animation must time the actions exactly, to produce a convincing result.

The "moving hold" animates between similar key frames, even characters sitting still can display some sort of movement, such as the torso moving in and out with breathing.

Slow In and Slow Out

The movement of the human body, and most other objects, needs time to accelerate and slow down. For this reason, animation looks more realistic if it has more drawings near the beginning and end of an action, emphasizing the extreme poses, and fewer in the middle. This principle goes for characters moving between two extreme poses, such as sitting down and standing up, but also for inanimate, moving objects, like the bouncing ball in the above illustration.

Arc

Most natural action tends to follow an arched trajectory, and animation should adhere to this principle by following implied "arcs" for greater realism. This technique can be applied to a moving limb by rotating a joint, or a thrown object moving along a parabolic trajectory. The exception is mechanical movement, which typically moves in straight lines.

As an object's speed or momentum increases, arcs tend to flatten out in moving ahead and broaden in turns. In baseball, a fastball would tend to move in a straighter line than other pitches; while a figure skater moving at top speed would be unable to turn as sharply as a slower skater, and would need to cover more ground to complete the turn.

An object in motion that moves out of its natural arc for no apparent reason will appear erratic rather than fluid. For example, when animating a pointing finger, the animator should be certain that in all drawings in between the two extreme poses, the fingertip follows a logical arc from one extreme to the next. Traditional animators tend to draw the arc in lightly on the paper for reference, to be erased later.

Secondary Action

Secondary Action: as the horse runs, its mane and tail follow the movement of the body.

Adding secondary actions to the main action gives a scene more life, and can help to support the main action. A person walking can simultaneously swing his arms or keep them in his pockets, speak or whistle, or express emotions through facial expressions. The important thing about secondary actions is that they emphasize, rather than take attention away from the main action. If the latter is the case, those actions are better left out. For example, during a dramatic movement, facial expressions will often go unnoticed. In these cases it is better to include them at the beginning and the end of the movement, rather than during.

Timing

Timing refers to the number of drawings or frames for a given action, which translates to the speed of the action on film. On a purely physical level, correct timing makes objects appear to obey the laws of physics; for instance, an object's weight determines how it reacts to an impetus, like a push. Timing is critical for establishing a character's mood, emotion, and reaction. It can also be a device to communicate aspects of a character's personality.

Exaggeration

Exaggeration is an effect especially useful for animation, as perfect imitation of reality can look static and dull in cartoons. The level of exaggeration depends on whether one seeks realism or a particular style, like a caricature or the style of a specific artist. The classical definition of exaggeration, employed by Disney, was to remain true to reality, just presenting it in a wilder, more extreme form.

Other forms of exaggeration can involve the supernatural or surreal, alterations in the physical features of a character; or elements in the storyline itself. It is important to employ a certain level of restraint when using exaggeration. If a scene contains several elements, there should be a balance in how those elements are exaggerated in relation to each other, to avoid confusing or overawing the viewer.

Solid Drawing

The principle of solid drawing means taking into account forms in three-dimensional space, or giving them volume and weight. The animator needs to be a skilled artist and has to understand the basics of three-dimensional shapes, anatomy, weight, balance, light and shadow, etc. For the classical animator, this involved taking art classes and doing sketches from life.

One thing in particular that Johnston and Thomas warned against was creating "twins": characters whose left and right sides mirrored each other, and looked lifeless. Modern-day computer animators draw less because of the facilities computers give them, yet their work benefits greatly from a basic understanding of animation principles, and their additions to basic computer animation.

Appeal

Appeal in a cartoon character corresponds to what would be called charisma in an actor. A character who is appealing is not necessarily sympathetic – villains or monsters can also be appealing – the important thing is that the viewer feels the character is real and interesting. There are several tricks for making a character connect better with the audience; for likable characters a symmetrical or particularly baby-like face tends to be effective. A complicated or hard to read face will lack appeal, it may more accurately be described as 'captivation' in the composition of the pose, or the character design.

Inverting Animations

Problem

You want to have an animation exactly mirroring another one that you just played.

Solution

Reverse the direction of the ClutterTimeline associated with the animation.

For example, here's how to invert an implicit animation which moves an actor along the x axis. The direction of the animation is inverted when the movement along the x axis is completed; it is also inverted if the mouse button is pressed on the actor.

First, set up the animation:

```
ClutterAnimation *animation;
/*
* animate actor to x = 300.0;
* the implicit animation functions return a ClutterAnimation
* which we can use to invert the timeline
*/
```

```
    animation = clutter_actor_animate (actor,
    CLUTTER_EASE_IN_OUT_CUBIC,
                                      2000,
                                      "x", 300.0,
                                      NULL);
    /* callback for when the animation completes */
    g_signal_connect (animation,
                      "completed",
                      G_CALLBACK (_animation_done_cb),
                      NULL);
    /*
    * callback for when the mouse button is pressed on the
actor;
    * note the animation is passed as user data, so we can
    * get at the timeline
    */
    g_signal_connect (actor,
                      "button-press-event",
                      G_CALLBACK (_on_click_cb),
                      animation);
```

Next, add a function for inverting the timeline:

```
    static void
    _invert_timeline (ClutterTimeline *timeline)
    {
    ClutterTimelineDirection        direction        =
clutter_timeline_get_direction (timeline);
    if (direction == CLUTTER_TIMELINE_FORWARD)
    direction = CLUTTER_TIMELINE_BACKWARD;
    else
    direction = CLUTTER_TIMELINE_FORWARD;
    clutter_timeline_set_direction (timeline, direction);
    }
```

Then add a function which calls _invert_timeline when the animation completes. More importantly, the callback should stop emission of the "completed" signal by the animation. This prevents the ClutterAnimation underlying the implicit animation from being unreferenced; which in turn allows it to be inverted:

```
    static void
    _animation_done_cb (ClutterAnimation *animation,
    gpointer user_data)
    {
    /* stop the completed signal before the ClutterAnimation
is unreferenced */
    g_signal_stop_emission_by_name (animation, "completed");
    /* invert the timeline associated with the animation */
    ClutterTimeline *timeline = clutter_animation_get_timeline
(animation);
    _invert_timeline (timeline);
    }
```

Finally, the click callback function uses the same _invert_timeline function if the animation is playing; but if the animation is stopped, it will start it instead:

```
    static void
    _on_click_cb (ClutterActor *actor,
                        ClutterEvent *event,
                        gpointer user_data)
    {
    ClutterAnimation *animation = (ClutterAnimation
*)user_data;
    ClutterTimeline *timeline = clutter_animation_get_timeline
(animation);
    if (clutter_timeline_is_playing (timeline))
                       {
    _invert_timeline (timeline);
                       }
    else
                       {
    clutter_timeline_start (timeline);
                       }
    }
```

Discussion

If you are using ClutterAnimator rather than implicit animations, clutter_animator_get_timeline() enables you to get the underlying timeline; you could then use the techniques shown above to invert it.

ClutterState enables a different approach to “inverting” an animation: rather than having a single animation which you invert, you would define two or more keys for an actor (or set of actors) and transition between them. For the example above, you would define two keys: one for the actor's initial position; and a second for the actor at x = 300.0. You would also define the transition between them: 2000 milliseconds with a CLUTTER_EASE_IN_

OUT_CUBIC easing mode.

With the states defined, you would then use clutter_state_set_state() inside callbacks to animate the actor between the two xpositions. Behind the scenes, ClutterState would handle the animations and timelines for you.

Fading an actor out of or into view

Problem

You want to animate an actor so that it fades out of or into view.

Solution

Animate the actor's opacity property.

You can do this using any of the approaches provided by the animation API. Here's how to fade out an actor (until it's completely transparent) using implicit animations:

```
/* fade out actor over 4000 milliseconds */
clutter_actor_animate (actor,
CLUTTER_EASE_OUT_CUBIC,
4000,
"opacity", 0,
NULL);
```

Here's an example of a rectangle fading out using this animation:

CLUTTER_EASE_OUT_CUBIC is one of the Clutter easing modes.

Here's an example of the transitions you could use to fade an actor in and out using ClutterState:

```
ClutterState *transitions = clutter_state_new ();
/* all transitions last for 2000 milliseconds */
clutter_state_set_duration (transitions, NULL, NULL, 2000);
/* transition from any state to "fade-out" state */
clutter_state_set (transitions,
NULL,/* from state (NULL means "any") */
"fade-out",/* to state */
```

```
    actor, "opacity", CLUTTER_EASE_OUT_QUAD, 0,
    NULL);
    /* transition from any state to "fade-in" state */
    clutter_state_set (transitions, NULL, "fade-in",
    actor, "opacity", CLUTTER_EASE_OUT_QUAD, 255,
    NULL);
    /* put the actor into the "fade-out" state with no animation
*/
    clutter_state_warp_to_state (transitions, "fade-out");
```

You would then trigger an animated state change as events occur in the application (*e.g.* mouse button clicks):

```
    clutter_state_set_state (transitions, "fade-in");
```

Here's an example of this animation fading in then out again:

Note

ClutterState is most useful where you need to animate an actor backwards and forwards between multiple states (*e.g.* fade an actor in and out of view). Where you just want to fade an actor in or out once, clutter_actor_animate() is adequate.

Discussion

Reducing an actor's transparency to zero does not make it inactive: the actor will still be reactive even if it's not visible (responding to key events, mouse clicks etc.).

To make it really "disappear", you could use clutter_actor_hide() once you'd made the actor fully transparent.

Purpose of Dialogue and Animation

Dialogue can be the trickiest part of a script. Lucky for us, Jean Ann Wright continues her series of articles on writing for television animation and this month tells us how to tame the words of animated actors.

When Batman kicks butt, no words are needed. Justice League and all related characters are trademarks.

The Purpose of Dialogue

When Batman kicks butt, he doesn't need a lot of dialogue to dump the dumbdumbs. At its best animation is all about action and movement; it explores space and time. You want to show, not tell, your story. There are cartoons with no dialogue at all! But three dialogue blocks per page and no more than three short sentences per block are normal. Generally, in animation dialogue should be used only after you've tried all other methods of communication. Silence can accompany discoveries, revelations and deep emotions. Dialogue is used to reveal the characters. It provides direction, moving the story along and advancing the plot. It discloses information. It provides conflict. And it sets the spirit or mood of the story, whether it's a comedy or drama.

Revealing Character

Sometimes only dialogue can expose the real motivations and secrets of a character in all their complexity. It's especially effective when it exposes the character in an entirely new way from what we as an audience expect. We use dialogue to establish relationships. Dialogue reflects feelings and attitudes.

Be sure you know your characters. Each character has his own agenda, often hidden. What is really being said? Which character is driving each scene? Your characters can be driving the action directly or indirectly. Direct dialogue drives people apart. Indirect dialogue draws people together. Characters may talk around a problem as we often do in real life. There may be subtext. But because younger kids probably won't understand subtlety, writing targeted at preschoolers should say what it means. Writing will also be more direct in shorter cartoons, as there simply is not time for many shadings.

A longer story digs deeper. To do this try using questions in order to get beneath the surface. Dialogue should never be interchangeable between characters. It should be dialogue that only that character would say. The words should be words that this character would use. Each character should have a different rhythm, perhaps a different sentence length. Dialogue reveals education, cultural and ethnic background, age. Use wording and colorful expressions that are individual to that one character. Unique phrases can serve as a character signature.

Moving the Story Along

A good animation story has to keep moving. Dialogue shouldn't slow it down. It should serve the plot. Dialogue is one way to tell the story, but the dialogue should always disclose tidbits that the characters must tell each other, not just information that you as a writer want the audience to know. Characters make discoveries about what's happening and discover secrets about each other.

Information and Conflict

All the exposition doesn't have to come right away. We want to know what happened before the story started that's motivating our characters now.

But information can come out throughout the story. Conflict in dialogue or tension between views is a good way to get information out and keep it interesting. Do be clear enough so that your young viewers understand, but don't say everything. Leave enough unsaid that the audience becomes involved and wants to know more.

The Mood of the Story

Set the tone of the story right away. This is especially important in comedy, so that we know that it's OK to laugh. The type of dialogue must be appropriate for the genre of that specific series.

Characteristics of Dialogue

Good dialogue has a beat, a rhythm, a melody. It's affected by time, place, the weather, etc. It's intangible, like mist, and it depends upon your characters and who they are, their relationships, the situation, the genre, the world of that series, the target age of your audience, the length of the script, and who you are as you're writing the dialogue. Keep it simple; less is more.

For young children keep the words simple enough that they'll understand.

Dialogue sounds like real talk, but it isn't. It's the essence of real talk with thematic content and an ongoing exchange of power. It must always be easily understandable and clear. You might want to repeat important story points, especially for preschoolers, but repeat with a twist.

Comedy Dialogue

The best comedy comes out of character. Be sure you have funny, exaggerated characters, reacting to a funny situation, and speaking in a funny way. Reactions are all-important to comedy. And so is timing! Try to avoid straight lines wherever you can. Use dialogue that plays off the situation.

If there's a fire, "Let's hot foot it out of here!" Then play the next line off of that. A straight man can serve as a foil for the one-liners. Insults can be funny. Comedy dialogue develops with a setup and then a surprise punch line. The punch line comes at the end of a speech. Comedy scenes usually go out on a laugh line (a button).

Fig. Homer Simpson's

Writing the Dialogue

If you can listen to tapes of your established characters in advance, do it. Your story should be set up in the first few words of dialogue. From the start, keep in mind your final end point, and build the dialogue towards the climax. Write less than you think you need. See and hear it as you write. Act it out in character. You'll want to add a new dimension with your dialogue, but don't make it so different that it doesn't sound like the established characters. Write the dialogue so that the actor can contribute something with his voice (a gulp, an excited squeal, a drawl). Think of Homer Simpson's "Doh!" Give your actors attitude, emotion, special phrasing. Character sneezes, etc. should be written with the dialogue so they're not missed during the recording session. If you're writing only one line for an incidental character, make that one line a jewel...really memorable. Keep your language appropriate for that series. If you're writing an original script, decide ahead of time whether you want your language up-to-date and fresh or classic for a longer shelf life for that show. Dialogue for children can be whimsical and full of contradictions and non-sense. Be original and clever!

6

Traditional Animation

Traditional animation, also referred to as classical animation, cel animation, or hand-drawn animation, is the oldest and historically the most popular form of animation. In a traditionally-animated cartoon, each frame is drawn by hand. The term "traditional animation" is often used in contrast with the now more commonly used computer animation.

Process

Storyboards

Traditionally-animated productions, just like other forms of animation, usually begin life as a *storyboard,* which is a script of sorts written with images as well as words, similar to a giant comic strip. The images allow the animation team to plan the flow of the plot and the composition of the imagery. The *storyboard artists* will have regular meetings with the director, and may have to redraw or "re-board" a sequence many times before it meets final approval.

Voice Recording

Before true animation begins, a preliminary soundtrack or "scratch track" is recorded, so that the animation may be more precisely synchronized to the soundtrack. Given the slow, methodical manner in which traditional animation is produced, it is almost always easier to synchronize animation to a pre-existing soundtrack than it is to synchronize a soundtrack to pre-existing animation. A completed cartoon soundtrack will feature music, sound effects, and dialogue performed by voice actors. However, the scratch track used during animation typically contains just the voices, any vocal songs that the characters must sing along to, and temporary musical score tracks; the final score and sound effects are added in post-production. In the case of most pre-1930 sound animated cartoons, the sound was *post-synched;* that is, the sound track was recorded after the film elements were finished by watching the film and performing the dialogue, music, and sound effects required. Some studios, most notably Fleischer Studios, continued to post-synch their cartoons through most of the 1930s, which allowed for the presence of the "muttered ad-libs" present in many *Popeye the Sailor* and *Betty Boop* cartoons.

Animatic

Often, an *animatic* or *story reel* is made after the soundtrack is created, but before full animation begins. An animatic typically consists of pictures of the storyboard synchronized with the soundtrack. This allows the animators and directors to work out any script and timing issues that may exist with the current storyboard. The storyboard and soundtrack are amended if necessary, and a new animatic may be created and reviewed with the director until the storyboard is perfected. Editing the film at the animatic stage prevents the animation of scenes that would be edited out of the film; as traditional animation is a very expensive and time-consuming process, creating scenes that will eventually be edited out of the completed cartoon is strictly avoided.

In the mid 1970s, these were known as videomatics and used primarily for test commercial projects. Advertising agencies today employ the use of animatics to test their commercials before they are made into full up spots. Animatics use drawn artwork, with moving pieces (for example, an arm that reaches for a product, or a head that turns). Video storyboards are similar to animatics, but do not have moving pieces. Photomatics are another option when creating test spots, but instead of using drawn artwork, there is a shoot in which hundreds of digital photographs are taken. The large amount of images to choose from may make the process of creating a test commercial a bit easier, as opposed to creating an animatic, because changes to drawn art take time and money. Photomatics generally cost more than animatics, as they require a shoot and on-camera talent.

Design and Timing

Once the animatic has been approved, it and the storyboards are sent to the design departments. Character designers prepare model sheets for all important characters and props in the film. These model sheets will show how a character or object looks from a variety of angles with a variety of poses and expressions, so that all artists working on the project can deliver consistent work. Sometimes, small statues known as *maquettes* may be produced, so that an animator can see what a character looks like in three dimensions. At the same time, the *background stylists* will do similar work for the settings and locations in the project, and the art directors and *colour stylists* will determine the art style and colour schemes to be used. While design is going on, the *timing director* (who in many cases will be the main director) takes the animatic and analyzes exactly what poses, drawings, and lip movements will be needed on what frames. An *exposure sheet* (or *X-sheet* for short) is created; this is a printed table that breaks down the action, dialogue, and sound frame-by-frame as a guide for the animators. If a film is based more strongly in music, a *bar sheet* may be prepared in addition to or instead of an X-sheet. Bar sheets show the relationship between the on-screen action, the dialogue, and the actual musical notation used in the score.

Layout

Layout begins after the designs are completed and approved by the director. The layout process is the same as the blocking out of shots by a cinematographer on a live-action film. It is here that the background layout artists determine the camera angles, camera paths, lighting, and shading of the scene. Character layout artists will determine the major poses for the characters in the scene, and will make a drawing to indicate each pose. For short films, character layouts are often the responsibility of the director. The layout drawings and storyboards are then spliced, along with the audio and an animatic is formed(not to be confused by its predecessor the leica reel).The term "animatic" was originally coined by Disney animation studios.

Animation

Once the Animatic is finally approved by the director, animation begins. In the traditional animation process, animators will begin by drawing sequences of animation on sheets of transparent paper perforated to fit the peg bars in their desks, often using coloured pencils, one picture or "frame" at a time. A *key animator* or *lead animator* will draw the key drawings in a scene, using the character layouts as a guide. The key animator draws enough of the frames to get across the major points of the action; in a sequence of a character jumping across a gap, the key animator may draw a frame of the character as he is about to leap, two or more frames as the character is flying through the air, and the frame for the character landing on the other side of the gap. Timing is important for the animators drawing these frames; each frame must match exactly what is going on in the soundtrack at the moment the frame will appear, or else the discrepancy between sound and visual will be distracting to the audience. For example, in high-budget productions, extensive effort is given in making sure a speaking character's mouth matches in shape the sound that character's actor is producing as he or she speaks. While working on a scene, a key animator will usually prepare a *pencil test* of the scene. A pencil test is a preliminary version of the final animated scene; the pencil drawings are quickly photographed or scanned and synced with the necessary soundtracks. This allows the animation to be reviewed and improved upon before passing the work on to his *assistant animators,* who will go add details and some of the missing frames in the scene. The work of the assistant animators is reviewed, pencil-tested, and corrected until the lead animator is ready to meet with the director and have his scene *sweatboxed*, or reviewed by the director, producer, and other key creative team members. Similar to the storyboarding stage, an animator may be required to re-do a scene many times before the director will approve it. In high-budget animated productions, often each major character will have an animator or group of animators solely dedicated to drawing that character.

The group will be made up of one supervising animator, a small group of key animators, and a larger group of assistant animators. For scenes where two characters interact, the key animators for both characters will decide which character is "leading" the scene, and that character will be drawn first. The second character will be animated to react to and support the actions of the "leading" character. Once the key animation is approved, the lead animator forwards the scene on to the *clean-up department,* made up of the *clean-up animators* and the *inbetweeners.* The clean-up animators take the lead and assistant animators' drawings and trace them onto a new sheet of paper, taking care in including all of the details present on the original model sheets, so that it appears that one person animated the entire film. The *inbetweeners* will draw in whatever frames are still missing *in between* the other animators' drawings. This procedure is called tweening. The resulting drawings are again pencil-tested and sweatboxed until they meet approval. At each stage during pencil animation, approved artwork is spliced into the Leica reel. This process is the same for both *character animation* and *special effects animation*, which on most high-budget productions are done in separate departments. *Effects animators* animate anything that moves and is not a character, including props, vehicles, machinery and phenomena such as fire, rain, and explosions.

Sometimes, instead of drawings, a number of special processes are used to produce special effects in animated films; rain, for example, has been created in Disney animated films since the late-1930s by filming slow-motion footage of water in front of a black background, with the resulting film superimposed over the animation.

Pencil Test

After all the drawings are cleaned-up, they are then photographed on an animation camera, usually on black, and white film stock. Nowadays, pencil tests can be made using a video camera, and computer software.

Backgrounds

While the animation is being done, the *background artists* will paint the sets over which the action of each animated sequence will take place. These backgrounds are generally done in gouache or acrylic paint, although some animated productions have used backgrounds done in watercolour, oil paint, or even crayon. Background artists follow very closely the work of the background layout artists and colour stylists (which is usually compiled into a workbook for their use), so that the resulting backgrounds are harmonious in tone with the character designs.

Traditional Ink-and-paint and Camera

When an entire sequence has been transferred to cels, the photography process begins. Each cel involved in a frame of a sequence is laid on top of each other, with the background at the bottom of the stack. A piece of glass is lowered onto the artwork in order to flatten any irregularities, and the composite image is then photographed by a special animation camera, also called rostrum camera. The cels are removed, and the process repeats for the next frame until each frame in the sequence has been photographed. Each cel has *registration holes*, small holes along the top or bottom edge of the cel, which allow the cel to be placed on corresponding peg bars before the camera to ensure that each cel aligns with the one before it; if the cels are not aligned in such a manner, the animation, when played at full speed, will appear "jittery." Sometimes, frames may need to be photographed more than once, in order to implement superimpositions and other camera effects. Pans are created by either moving the cels or backgrounds one step at a time over a succession of frames (the camera does not pan; it only zooms in and out). As the scenes come out of final photography, they are spliced into the Leica reel, taking the place of the pencil animation. Once every sequence in the production has been photographed, the final film is sent for development and processing, while the final music and sound effects are added to the soundtrack. Again, editing in the traditional live-action sense is generally not done in animation, but if it is required it is done at this time, before the final print of the film is ready for duplication or broadcast. Among the most common types of animation rostrum cameras was the Oxberry. Such cameras were always made of black anodized aluminum, and commonly had 2 pegbars, one at the top and one at the bottom of the lightbox. The Oxberry Master Series had four pegbars, two above and two below, and sometimes used a "floating pegbar" as well. The height of the column on which the camera was mounted determined the amount of zoom achievable on a piece of artwork. Such cameras were massive mechanical affairs which might weigh close to a ton and take hours to break down or set up. In the later years of the animation rostrum camera, stepper motors controlled by computers were attached to the various axes of movement of the camera, thus saving many hours of hand cranking by human operators. A notable early use of computer cameras was in *Star Wars* (1977), using the Dykstra system at Lucas' Sun Valley facility. Gradually, motion control techniques were adopted throughout the industry. While several computer camera software packages became available in the early 1980s, the Tondreau System became one of the most widely adopted. Digital ink and paint processes gradually made these traditional animation techniques and equipment obsolete.

Digital Ink and Paint

The current process, termed "digital ink and paint," is the same as traditional ink and paint until after the animation drawings are completed; instead of being transferred to cels, the animators' drawings are scanned into a computer, where they are coloured and processed using one or more of a variety of software packages. The resulting drawings are composited in the computer over their respective backgrounds, which have also been scanned into the computer (if not digitally painted), and the computer outputs the final film by either exporting a digital video file, using a video cassette recorder, or printing to film using a high-resolution output device.

Use of computers allows for easier exchange of artwork between departments, studios, and even countries and continents (in most low-budget animated productions, the bulk of the animation is actually done by animators working in other countries, including South Korea, Japan, Singapore, Mexico, and India). The last major feature film to use traditional ink and paint was Studio Ghibli's *Princess Mononoke* (1997); the last major animation production to use the traditional process is Cartoon Network's series *Ed Edd n Eddy* (1999–2009), although it was forced to switch to digital paint in 2004. Minor productions such as Hair High (2004) by Bill Plympton have used traditional cels long after the introduction of digital techniques.

Digital ink and paint has been in use at Walt Disney Feature Animation since 1989, where it was used for the final rainbow shot in *The Little Mermaid.* All subsequent Disney animated features were digitally inked-and-painted, using Disney's proprietary CAPS (Computer Animation Production System) technology, developed primarily by Pixar (the last Disney feature using CAPS was *Home on the Range*). Most other studios use one of a number of other high-end software packages such as Toon Boom Harmony, Toonz, Animo, and even consumer-level applications such as Adobe Flash, Toon Boom Studio and TVPaint.

Computers and Digital Video Cameras

Computers and digital video cameras can also be used as tools in traditional cel animation without affecting the film directly, assisting the animators in their work and making the whole process faster and easier. Doing the layouts on a computer is much more effective than doing it by traditional methods. Additionally, video cameras give the opportunity to see a "preview" of the scenes and how they will look when finished, enabling the animators to correct and improve upon them without having to complete them first. This can be considered a digital form of *pencil testing*.

Techniques

The Cel & Limited Animation

The cel is an important innovation to traditional animation, as it allows some parts of each frame to be repeated from frame to frame, thus saving labor. A simple example would be a scene with two characters on screen, one of which is talking and the other standing silently. Since the latter character is not moving, it can be displayed in this scene using only one drawing, on one cel, while multiple drawings on multiple cels will be used to animate the speaking character. For a more complex example, consider a sequence in which a girl sets a plate upon a table. The table will stay still for the entire sequence, so it can be drawn as part of the background. The plate can be drawn along with the character as the character places it on the table. However, after the plate is on the table, the plate will no longer move, although the girl will continue to move as she draws her arm away from the plate. In this example, after the girl puts the plate down, the plate can then be drawn on a separate cel from the girl.

Further frames will feature new cels of the girl, but the plate does not have to be redrawn as it is not moving; the same cel of the plate can be used in each remaining frame that it is still upon the table. The cel paints were actually manufactured in shaded versions of each colour to compensate for the extra layer of cel added between the image and the camera, in this example the still plate would be painted slightly brighter to compensate for being moved one layer down. In very early cartoons made before the use of the cel, such as *Gertie the Dinosaur* (1914), the entire frame, including the background and all characters and items, were drawn on a single sheet of paper, then photographed. Everything had to be redrawn for each frame containing movements. This led to a "jittery" appearance; imagine seeing a sequence of drawings of a mountain, each one slightly different from the one preceding it.

The pre-cel animation was later improved by using techniques like the slash and tear system invented by Raoul Barre; the background and the animated objects were drawn on separate papers. A frame was made by removing all the blank parts of the papers where the objects were drawn before being placed on top of the backgrounds and finally photographed. The cel animation process was invented by Earl Hurd and John Bray in 1915. In lower-budget productions, this "shortcut" is used in a greater capacity.

For example, in a scene in which a man is sitting in a chair and talking, the chair and the body of the man may be the same in every frame; only his head is redrawn, or perhaps even his head stays the same while only his mouth moves. This is known as *limited animation.* The process was popularized in theatrical cartoons by United Productions of America and used in most television animation, especially that of Hanna-Barbera. The end result does not look very lifelike, but is inexpensive to produce, and therefore allows cartoons to be made on small television budgets.

"Shooting on Twos"

Moving characters are often shot "on twos", that is to say, one drawing is shown for every two frames of film (which usually runs at 24 frames per second), meaning there are only 12 drawings per second. Even though the image update rate is low, the fluidity is satisfactory for most subjects. However, when a character is required to perform a quick movement, it is usually necessary to revert to animating "on ones", as "twos" are too slow to convey the motion adequately. A blend of the two techniques keeps the eye fooled without unnecessary production cost. Animation for television is usually produced on tight budgets. In addition to the use of limited animation techniques, television animation may be shot on "threes", or even "fours", i.e. three or four frames per drawing. This translates to only eight or six drawings per second.

Animation Loops

Creating *animation loops* or *animation cycles* is a labor-saving technique for animating repetitive motions, such as a character walking or a breeze blowing through the trees. In the case of walking, the character is animated taking a step with his right foot, then a step with his left foot. The loop is created so that, when the sequence repeats, the motion is seamless. However, since an animation loop essentially uses the same bit of animation over and over again, it is easily detected and can in fact become distracting to an audience. In general, they are used only sparingly by productions with moderate or high budgets. Ryan Larkin's 1969 Academy Award nominated National Film Board of Canada short *Walking* makes creative use of loops. In addition, a promotional music video featuring the Soul Coughing song "Circles" poked fun at animation loops as they are often seen in *The Flintstones,* in which Fred and Barney, supposedly walking in a house, wonder why they keep passing the same table and vase over and over again.

Multiplane Camera

The multiplane camera is a tool used to add depth to scenes in 2D animated movies, called the multiplane effect or the parallax process. The art is placed on different layers of glass plates, and as the camera moves vertically towards or away from the artwork levels, the camera's viewpoint appears to move through the various layers of artwork in 3D space. The panorama views in *Pinocchio* are examples of the effects a multiplane camera can achieve. Different versions of the camera have been made through time, but the most famous is the one developed by the Walt Disney studio beginning with their 1937 short *The Old Mill.* Another one, the "Tabletop", was developed by Fleischer Studios. The Tabletop, first used in 1934's *Poor Cinderella*, used miniature sets made of paper cutouts placed in front of the camera on a rotating platform, with the cels between them. By rotating the entire setup one frame at a time in accordance with the cel animation, realistic panoramas could be created. Ub Iwerks and Don Bluth also built multiplane cameras for their studios.

Xerography

Applied to animation by Ub Iwerks at the Walt Disney studio during the late 1950s, the electrostatic copying technique called xerography allowed the drawings to be copied directly onto the cels, eliminating much of the "inking" portion of the ink-and-paint process. This saved time and money, and it also made it possible to put in more details and to control the size of the xeroxed objects and characters (this replaced the little known, and seldom used, photographic lines technique at Disney, used to reduce the size of animation when needed). At first it resulted in a more sketchy look, but the technique was improved upon over time. The xerographic method was first tested by Disney in a few scenes of *Sleeping Beauty*, and was first fully used in the short film *Goliath II*, while the first feature entirely using this process was *One Hundred and One Dalmatians* (1961). The graphic style of this film was strongly influenced by the process. Some hand inking was still used together with xerography in this and subsequent films when distinct coloured lines were needed. Later, coloured toners became available, and several distinct line colours could be used, even simultaneously. For instance, in *The Rescuers* the characters outlines are gray. White and blue toners were used for special effects, such as snow and water.

APT Process

Invented by Dave Spencer for the 1985 Disney film *The Black Cauldron*, the APT (Animation Photo Transfer) process was a technique for transferring the animators' art onto cels. Basically, the process was a modification of a repro-photographic process; the artists' work were photographed on high-contrast "litho" film, and the image on the resulting negative was then transferred to a cel covered with a layer of light sensitive dye. The cel was exposed through the negative. Chemicals were then used to remove the unexposed portion. Small and delicate details were still inked by hand if needed. Spencer received an Academy Award for Technical Achievement for developing this process.

Cel Overlay

A *cel overlay* is a cel with inanimate objects used to give the impression of a foreground when laid on top of a ready frame. This creates the illusion of depth, but not as much as a multiplane camera would. A special version of cel overlay is called *line overlay*, made to complete the background instead of making the foreground, and was invented to deal with the sketchy appearance of xeroxed drawings. The background was first painted as shapes and figures in flat colours, containing rather few details. Next, a cel with detailed black lines was laid directly over it, each line drawn to add more information to the underlying shape or figure and give the background the complexity it needed. In this way, the visual style of the background will match that of the xeroxed character cels. As the xerographic process evolved, line overlay was left behind.

Computers and Traditional Animation

The methods mentioned above describe the techniques of an animation process that originally depended on cels in its final stages, but painted cels are rare today as the computer moves into the animation studio, and the outline drawings are usually scanned into the computer and filled with digital paint instead of being transferred to cels and then coloured by hand. The drawings are composited in a computer programme on many transparent "layers" much the same way as they are with cels, and made into a sequence of images which may then be transferred onto film or converted to a digital video format. It is now also possible for animators to draw directly into a computer using a graphics tablet, Cintiq or a similar device, where the outline drawings are done in a similar manner as they would be on paper. The Goofy short *How To Hook Up Your Home Theater* (2007) represented Disney's first project based on the paperless technology available today.

Some of the advantages are the possibility and potential of controlling the size of the drawings while working on them, drawing directly on a multiplane background and eliminating the need of photographing line tests and scanning. Though traditional animation is now commonly done with computers, it is important to differentiate computer-assisted traditional animation from 3D computer animation, such as *Toy Story* and *ReBoot.* However, often traditional animation and 3D computer animation will be used together, as in Don Bluth's *Titan A.E.* and Disney's *Tarzan* and *Treasure Planet.* Most anime still use traditional animation today. DreamWorks executive Jeffrey Katzenberg coined the term "tradigital animation" to describe films produced by his studio which incorporated elements of traditional and computer animation equally, such as *Spirit: Stallion of the Cimarron* and *Sinbad: Legend of the Seven Seas*. Interestingly, many modern video games such as *Viewtiful Joe, The Legend of Zelda: The Wind Waker* and others use "cel-shading" animation filters to make their full 3D animation appear as though it were drawn in a traditional cel style. This technique was also used in the animated movie *Appleseed,* and cel-shaded 3D animation is typically integrated with cel animation in Disney films and in many television shows, such as the Fox animated series *Futurama.*

Rotoscoping

Rotoscoping is a method of traditional animation invented by Max Fleischer in 1915, in which animation is "traced" over actual film footage of actors and scenery. Traditionally, the live action will be printed out frame by frame and registered. Another piece of paper is then placed over the live action printouts and the action is traced frame by frame using a lightbox. The end result still looks hand drawn but the motion will be remarkably lifelike. *Waking Life* is a full-length, rotoscoped animated movie, as is *American Pop* by Ralph Bakshi. The popular music video for A-ha's song "Take On Me" also featured rotoscoped animation, along with live action. In most cases, rotoscoping is mainly used to aid the animation of realistically rendered human beings, as in *Snow White and the Seven Dwarfs, Peter Pan,* and *Sleeping Beauty*. A method related to conventional rotoscoping was later invented for the animation of solid inanimate objects, such as cars, boats, or doors. A small live action model of the required object was built and painted white, while the edges of the model were painted with thin black lines. The object was then filmed as required for the animated scene by moving the model, the camera, or a combination of both, in real time or using stop-motion animation. The film frames were then printed on paper, showing a model made up of the painted black lines.

After the artists had added details to the object not present in the live-action photography of the model, it was xeroxed onto cels. A notable example is Cruella de Vil's car in Disney's *One Hundred and One Dalmatians*. The process of transferring 3D objects to cels was greatly improved in the 1980s when computer graphics advanced enough to allow the creation of 3D computer generated objects that could be manipulated in any way the animators wanted, and then printed asoutlines on paper before being copied onto cels using Xerography or the APT process. This technique was used in Disney films such as *Oliver and Company* (1988) and *The Little Mermaid* (1989). This process has more or less been superseded by the use of cel-shading. Related to rotoscoping are the methods of vectorizing live-action footage, in order to achieve a very graphical look, like in Richard Linklater's film *A Scanner Darkly*; and motion-capturing actor's movements to use the data in 3D-animation, as in Robert Zemeckis's 2004 film *The Polar Express*.

Live-action Hybrids

Similar to the computer animation and traditional animation hybrids described above, occasionally a production will marry both live-action and animated footage. The live-action parts of these productions are usually filmed first, the actors pretending that they are interacting with the animated characters, props, or scenery; animation will then be added into the footage later to make it appear as if it has always been there. Like rotoscoping, this method is rarely used, but when it is, it can be done to terrific effect, immersing the audience in a fantasy world where humans and cartoons co-exist. Early examples include the silent *Out of the Inkwell* (begun in 1919) cartoons by Max Fleischer and Walt Disney's *Alice Comedies* (begun in 1923). Live-action and animation were later combined to successful effect in features such as *The Three Caballeros* (1944), *Anchors Aweigh* (1945), *Song of the South* (1946), *Mary Poppins* (1964), *Bedknobs and Broomsticks* (1971), *Heavy Traffic* (1973), *Coonskin* (1975) *Pete's Dragon* (1977), *Who Framed Roger Rabbit* (1988), *Rock-a-Doodle* (1992), *Cool World* (1992), *The Pagemaster* (1994) *Space Jam* (1996), and *Looney Tunes: Back In Action* (2003). Other significant live-action hybrids include the music video for Paula Abdul's hit song "Opposites Attract" and numerous television commercials, including those for cereals such as Frosted Flakes, Honey Nut Cheerios, Trix, and Rice Krispies.

Special Effects Animation

Besides traditional animated characters, objects and backgrounds, many other techniques are used to create special elements such as smoke, lightning and "magic", and to give the animation in general a distinct visual appearance. Notable examples can be found in movies such as *Fantasia*, *Wizards*, *The Lord of the Rings*, *The Little Mermaid*, *The*

Secret of NIMH and *The Thief and the Cobbler.* Today the special effects are mostly done with computers, but earlier they had to be done by hand. To produce these effects, the animators used different techniques, such as drybrush, airbrush, charcoal, grease pencil, backlit animation or, during shooting, the cameraman used multiple exposures with diffusing screens, filters or gels. For instance, the *Nutcracker Suite* segment in *Fantasia* has a fairy sequence where stippled cels are used, creating a soft pastel look.

Limited Animation

Limited animation is a process of making animated cartoons that does not redraw entire frames but variably reuses common parts between frames. One of its major trademarks is the stylized design in all forms and shapes, which in the early days was referred to as modern design. The short cartoons and feature films of Walt Disney from the 1930s and 1940s are widely acclaimed for depicting animated simulations of reality, with exquisite detail in every frame. However, this style of animation is very time-consuming and expensive. "Limited" animation creates an image that uses abstract art, symbolism, and fewer drawings to create the same effect, but at a much lower production cost. This style of animation depends upon animators' skill in emulating change without additional drawings; improper use of limited animation can be easily recognized as unnatural. It also encourages the animators to indulge in artistic styles that are not necessarily bound to the limits of the real world. The result is a new artistic style that could not have developed if animation was solely devoted to producing simulations of reality. Without limited animation, such ground-breaking films as *Yellow Submarine*, Chuck Jones' *The Dot and the Line*, and many others could never have been produced. The process of limited animation mainly aims at reducing the overall number of drawings. Film is projected at 24 frames per second. For movements in normal speed, most animation in general is done "on twos," meaning 12 drawings per second are recorded meaning that each drawing uses two frames of film. Faster movements may demand animation "on ones," while characters that do not move may be done with a single drawing (a "hold") for a certain amount of time. It is said that the Disney average was about 18 drawings per second, pretending that all characters of a scene share the same sheet of paper. Limited animation mainly reduces the number of inbetweens, the drawings between the keyframes which define a movement, and can cause stuttering if inbetweens are poorly setup. Overall, the use of limited animation does not necessarily imply lower quality as it allows the use of many timesaving techniques that can improve the quality and flow of the keyframes and overall presentation of an animation.

History

The use of budget-cutting animation measures in animation dates at least to the 1930s; a handful of the Bosko cartoons in the early years of the *Looney Tunes* series used several visible tricks to give the shorts the comparable appearance of the Disney shorts of the same era, even though they were produced on a budget of just over half of their Disney counterparts. The 1942 *Merrie Melodies* short "The Dover Boys" was a particular early prototype of the use of limited animation, though pressure from Warner

Bros. curtailed much further use of the technique. Limited animation was originally founded as an artistic device, though it was soon used widely as a cost-cutting measure rather than an aesthetic method. The UPA studio made the first serious effort to abandon the keyframe heavy approach perfected by Disney. Their first effort at limited animation, *Gerald McBoing-Boing*, won an Oscar, and it provided the impetus for this animation method to be accepted at the major Hollywood cartoon studios, including Warner Brothers and MGM. However, the real attraction of limited animation was the reduction in costs: because limited animation does not require as many drawings as fully keyframed animations, it is much less expensive to produce. The 1950s saw all of the major cartoon studios change their style to limited animation, to the point where painstaking detail in animation occurred only rarely. Limited animation techniques in America were used during the 1960s and 1970s to produce a great number of inexpensive Saturday morning cartoons. Such TV series as *Clutch Cargo* are known for being produced on extremely low budgets, with camera tricks used in place of actual animation. Despite the low quality of the animation, the TV cartoon studios Hanna-Barbera, Jay Ward and Filmation thrived during this period. The desire of the time to emulate full animation with limited animation led to many highly apparent visual issues.

Techniques

These techniques used to produce cartoons on a reduced budget included:

- cels and sequences of cels were used repeatedly — animators only had to draw a character walking once.

- characters are split up into different levels: only portions of a character, such as the mouth or an arm, would be animated on top of a static cel.
- clever choice of camera angles and editing.
- use of camera techniques such as panning to suggest movement. A famous implementation of this is the "crash" technique, which involves the camera shaking rapidly back and forth to simulate a shock wave.
- "smear animation:" movement is rapid and portrayed in only three frames: the beginning state, the ending state, and a "blur" frame similar to that of a picture taken with a camera that had a low shutter speed.
- cel reversal (simply using a mirror image of the cell to represent the opposite angle). Many cartoon characters are drawn symmetrically to expedite this technique.
- the visual elements were made subsidiary to audio elements, so that verbal humor and voice talent became more important factors for success ("talking heads").
- silhouette helped avoid having to keep track of shading on an animated character or object.
- sliding a cel across a background to suggest movement.
- Stock footage: sequences that are reused frequently. This is the case of the character transformations in the Magical girls subgenre of Japanese anime series. Filmation used this strategy for much of its productions.
- extensive recaps of previous episodes or segments, to cut down on the amount of new material necessary (used often in serials).

- The most egregious case of limited animation, known as Syncro-Vox, involved pasting a film of the moving lips of a real-life person over a still frame of an “animated” character to give the appearance that the character is doing the talking. Cambria Studios held a patent on the technology, and as such, it was primarily used on their productions, such as *Clutch Cargo*.

Examples

Animated cartoons which made use of limited animation include Gerald McBoing-Boing, Rooty Toot Toot, Mister Magoo, The Rocky and Bullwinkle Show, The Pink Panther, Clutch Cargo, and Kinnikuman. In recent years, nostalgia for the 1970s, combined with technologies such as Adobe Flash, have led to a revival of the genre of limited animation. Also, some modern graphic styles naturally translate into limited animation. Much of Japanese animation (anime) makes use of techniques adapted from limited animation. Osamu Tezuka started to use this technique in *Astro Boy* in order to save money and time. However, the technique is now combined with manga styles and aesthetics, and is a very distinct style.

Limited animation in anime is frequently used in action scenes such as mecha battles or transformation scenes. Limited animation is seen most frequently in television serials, but the aesthetic is so grounded in the medium that even bigger-budget feature films make use of it. Most Japanese animation is significantly less expensive than its American counterparts as a result, with Hayao Miyazaki’s

Ponyo (the most expensive Japanese animated feature film yet produced) costing only $34,000,000.

Rotoscoping

Rotoscoping is an animation technique in which animators trace over live-action film movement, frame by frame, for use in animated films. Originally, recorded live-action film images were projected onto a frosted glass panel and re-drawn by an animator. This projection equipment is called a rotoscope, although this device has been replaced by computers in recent years. In the visual effects industry, the term rotoscoping refers to the technique of manually creating a matte for an element on a live-action plate so it may be composited over another background.

History

The technique was invented by Max Fleischer, who used it in his series *Out of the Inkwell* starting around 1915, with his brother Dave Fleischer dressed in a clown outfit as the live-film reference for the character Koko the Clown. Max patented the method in 1917. Fleischer used rotoscoping in a number of his later cartoons, most notably the Cab Calloway dance routines in three Betty Boop cartoons from the early 1930s, and the animation of Gulliver in *Gulliver's Travels* (1939). The Fleischer studio's most effective use of rotoscoping was in their series of action-oriented *Superman cartoons*, in which Superman and the other animated figures displayed very realistic movement. Leon Schlesinger Productions, which produced the *Looney Tunes* and *Merrie Melodies* for Warner Bros., producing cartoons geared more towards exaggerated comedy, used rotoscoping only occasionally. Walt Disney and his animators employed it in *Snow White and the Seven Dwarfs* in 1937. Rotoscoping was also used in many of Disney's subsequent animated feature films with human characters, such as *Cinderella* in 1950. From the latter film onwards, the rotoscope was used mainly for studying human and animal motion, rather than actual tracing. Rotoscoping was used extensively in China's first animated feature film, *Princess Iron Fan* (1941), which was released under very difficult conditions during the Second Sino-Japanese War and World War II. It was used extensively in the Soviet Union, where it was known as "Éclair", from the late 1930s to the 1950s; its historical use was enforced as a realization of Socialist Realism. Most of the films produced with it were adaptations of folk tales or poems - for example, *The Night Before Christmas* or *The Tale of the Fisherman and the Fish.* Only in the early 1960s, after the Khrushchev Thaw, did animators start to explore very different aesthetics. The film crew on the Beatles animated film *Yellow Submarine* employed rotoscoping in numerous instances, most notably the sequence for "Lucy in the Sky with Diamonds." Ralph Bakshi used the technique quite extensively in his animated movies *Wizards* (1977), *The Lord of the Rings* (1978), *American Pop* (1981), and *Fire and Ice* (1983). Bakshi first turned to rotoscoping because he was refused by 20th Century Fox for a $50,000 budget increase to finish *Wizards*, and thus had to resort to the rotoscope technique to finish the battle sequences. Rotoscoping was also used in *Heavy Metal* (1981), three of a-ha's music videos, "Take on Me" (1985), "The Sun Always Shines on T.V." (1985), and "Train of Thought" (1986), and

Don Bluth's *Titan A.E.* (2000). While rotoscoping is generally known to bring a sense of realism to larger budget animated films, the American animation company Filmation, known for its budget-cutting limited TV animation, was also notable for its heavy usage of rotoscope to good effect in series such as *Flash Gordon*, *Blackstar*, and *He-Man and the Masters of the Universe*. Smoking Car Productions invented a digital rotoscoping process in 1994 for the creation of its critically-acclaimed adventure video game, *The Last Express*. The process was awarded U.S. Patent 6,061,462, *Digital Cartoon and Animation Process*.

In the mid-1990s, Bob Sabiston, an animator and computer scientist veteran of the MIT Media Lab, developed a computer-assisted "interpolated rotoscoping" process which he used to make his award-winning short film "Snack and Drink." Director Richard Linklater subsequently employed Sabiston's artistry and his proprietary Rotoshop software in the full-length feature films *Waking Life* (2001) and *A Scanner Darkly* (2006). Linklater licensed the same proprietary rotoscoping process for the look of both films. Linklater is the first director to use digital rotoscoping to create an entire feature film. Additionally, a 2005-08 advertising campaign by Charles Schwab uses Sabiston's rotoscoping work for a series of television spots, under the tagline "Talk to Chuck."

Technique

Rotoscope output can have slight deviations from the true line that differ from frame to frame, which when animated cause the animated line to shake unnaturally, or "boil". Avoiding boiling requires considerable skill in the person performing the tracing, though causing the "boil" intentionally is a stylistic technique sometimes used to emphasize the surreal quality of rotoscoping, as in the music video "Take on Me" and animated TV series *Delta State*. Rotoscoping (often abbreviated as "roto") has often been used as a tool for visual effects in live-action movies. By tracing an object, a silhouette (called a matte) is created that can be used to extract that object from a scene for use on a different background. While blue and green screen techniques have made the process of layering subjects in scenes easier, rotoscoping still plays a large role in the production of visual effects imagery.

Rotoscoping in the digital domain is often aided by motion tracking and onion-skinning software. Rotoscoping is often used in the preparation of garbage mattes for other matte-pulling processes. Rotoscoping has also been used to allow a special visual effect (such as a glow, for example) to be guided by the matte or rotoscoped line. One classic use of traditional rotoscoping was in the original three *Star Wars* films, where it was used to create the glowing lightsaber effect, by creating a matte based on sticks held by the actors. To achieve this, editors traced a line over each frame with the prop, then enlarged each line and added the glow.