Design and Analysis Techniques

Emil Hudson

DESIGN AND ANALYSIS TECHNIQUES

DESIGN AND ANALYSIS TECHNIQUES

Emil Hudson

BIBLIOTEX
Digital Library

Design and Analysis Techniques
by Emil Hudson

www.bibliotex.com

Ebook ISBN: 9781984664204

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

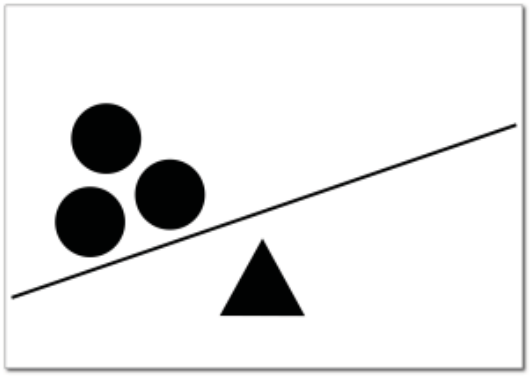
Contents

1

Principles of Design

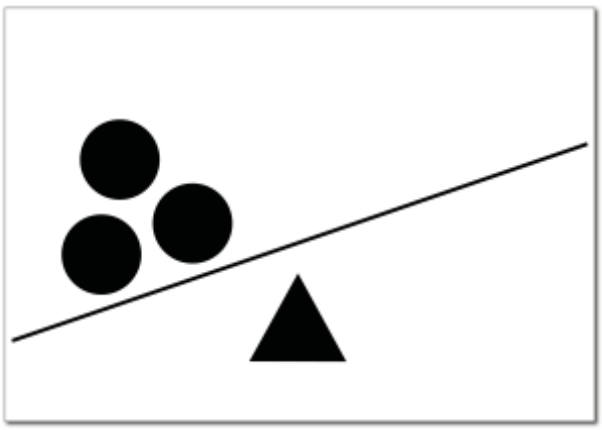
Generally, all the principles of design, also known as principles of composition, apply to any piece you may create. How you apply those principles determines how effective your design is in conveying the desired message and how attractive it appears. There is seldom only one correct way to apply each principle but check your documents to see how well you have applied each of these six principles of design.

BALANCE



Generally, all the principles of design, also known as principles of composition, apply to any piece you may create. How you apply those principles determines how effective your design is in conveying the desired message and how attractive it appears. There is seldom only one correct way to apply each principle but check your documents to see how well you have applied each of these six principles of design.

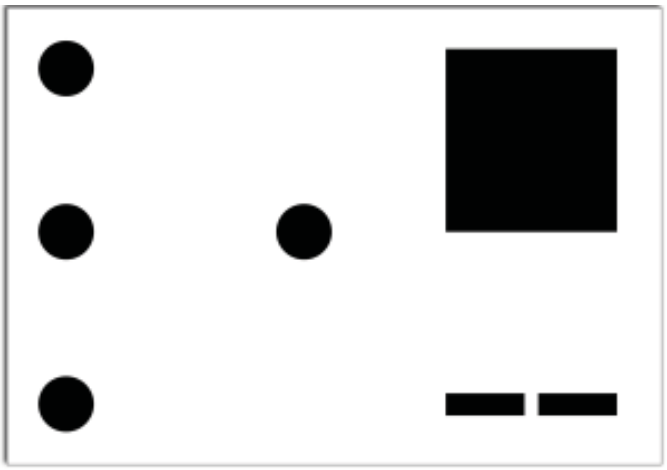
BALANCE



Visual balance comes from arranging elements on the page so that no one section is heavier than the other. Or, a designer may intentionally throw elements out of balance to create tension or a certain mood. Are your page elements all over the place are does each portion of the page balance out the rest? If out of balance, is it done purposely and with a specific intention in mind?

PROXIMITY/UNITY

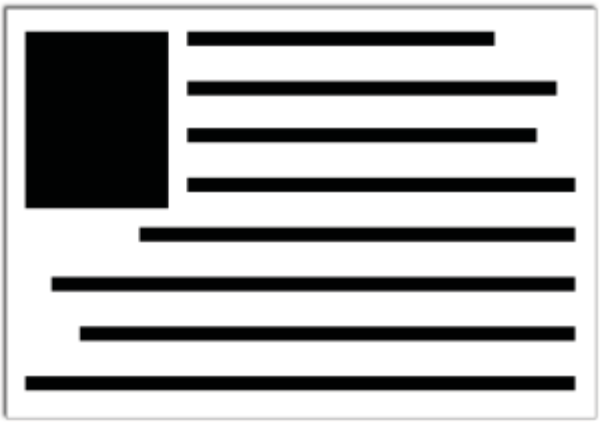
In design, proximity or closeness creates a bond between people and between elements on a page. How close together or far apart elements are placed suggests a relationship (or lack of) between otherwise disparate parts.



Unity is also achieved by using a third element to connect distant parts. Are title elements together? Is contact information all in one place? Do frames and boxes tie together or separate related elements in your document?

ALIGNMENT

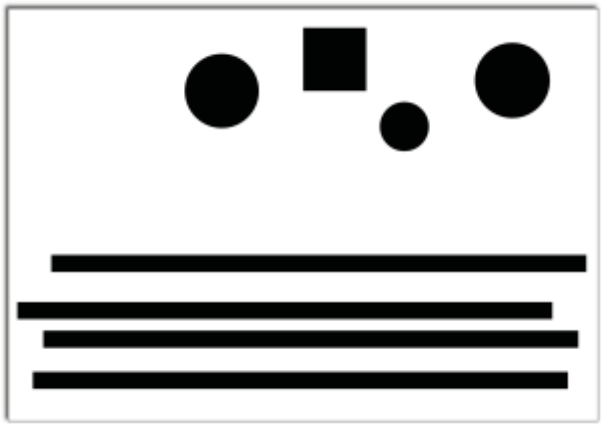
Alignment brings order to chaos. How you align type and graphics on a page and in relation to each other can make your layout easier or more difficult to read, foster familiarity, or bring excitement to a stale design. Have you used a grid?



Is there a common alignment — top, bottom, left, right, or centered — between blocks of text and graphics on the

page? Does your text alignment aid or hinder readability? If certain elements are out of alignment, was it done purposefully with a specific design goal in mind?

REPETITION/CONSISTENCY

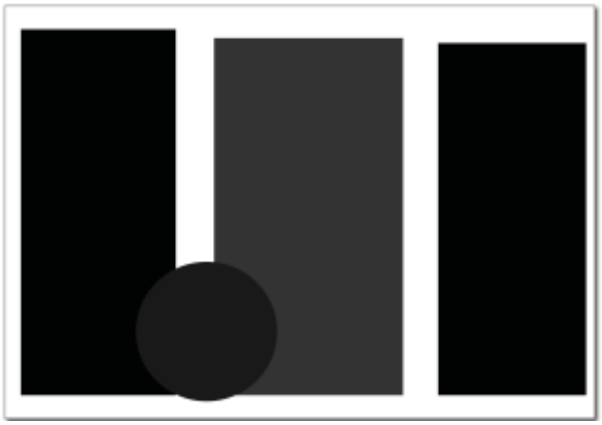


Repeating design elements and consistent use of type and graphics styles within a document shows a reader where to go and helps them navigate your designs and layouts safely. Insure that your document utilizes the principles of repetition, consistency, and unity in page design. Do page numbers appear in the same location from page to page? Are major and minor headlines consistent in size, style, or placement? Have you used a consistent graphic or illustration style throughout?

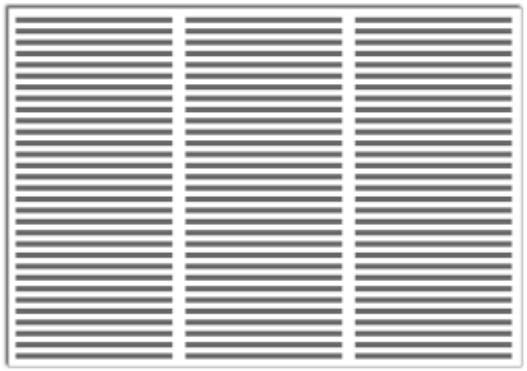
CONTRAST

In design, big and small elements, black and white text, squares and circles, can all create contrast in design. Contrast helps different design elements stand out. Is there enough contrast between your text (size and color) and background (color and pattern) to keep text readable? Is

everthing all the same size even when some elements are more important than others?



WHITE SPACE



Designs that try to cram too much text and graphics onto the page are uncomfortable and may be impossible to read. White space gives your design breathing room. Do you have enough space between columns of text? Does text run into frames or graphics? Do you have a generous margin? You can also have too much white space if items float on the page without any anchor.

ADDITIONAL PRINCIPLES OF DESIGN

In addition to or in place of some of these principles of design, other designers and instructors may include

principles such as harmony, flow, or hierarchy. Some principles may be combined or go by other names such as grouping (proximity), emphasis (use of various other principles to create a focal point). These are really different ways of expressing the same basic good page layout practices.

BASIC PRINCIPLES OF GRAPHIC DESIGN YOU TAKE FOR GRANTED EVERYDAY

Whether you're designing a logo, an event announcement, a social network banner, a letterhead, or an email newsletter; you absolutely need to know five basic principles of graphic design. Graphic designer and best selling author Robin Williams explains these principles in her classic book, *The Non-Designer's Design Book*.

Today we will be providing an overview of these principles using a few contemporary examples.

PROXIMITY

Proximity means grouping elements together so that you guide the viewer/reader to different parts of the message. Notice below in the template on the left, taken from Apple's Pages, related elements are grouped together, as opposed to the linear arrangement of amateur designs as shown on the right.

Though at first the elements may appear scattered, their proximity adds unity and continuity to the page. Even if you intend on sticking to templates, it still helps to know design principles for the purposes of customizing an existing design.

ALIGNMENT

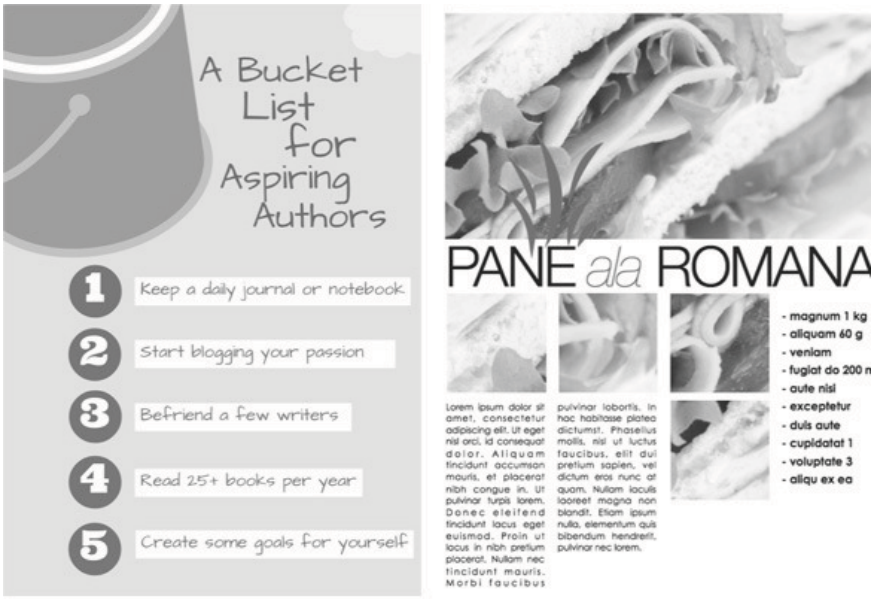
Another important design principle is aligning elements in a visual and readable arrangement. Most amateur designers start off by aligning everything in the center of the page, but that's not the only way. Again with the "scattered" looking design, we can see the alignment of elements that helps keep the design balanced. The top group of text is left-aligned, and three larger text elements are vertically aligned.

It's important to be consistent in the alignment of elements. When you look at the design and something doesn't feel right, play around with the alignment and see if the design can be improved.

REPETITION

Like the use of repetitious hooks in a song, repeating elements in a graphic design can be visually appealing. In

the two examples below, a numbered list is used, but there's also the repetition of the blue circles that make a bolder statement.



In the layout on the right, the image of the sandwiched is cropped and masked in repeating squares, as well as the use of repeated red strokes above the word "PANE." Repetition puts emphasis on particular elements of a design, and it draws the reader's attention to those elements.

CONTRAST

Contrast between design elements can make a presentation stand out and get noticed. Take for example this original template from the personal graphic design site, Canva.com. The elements of the design are grouped together, with strong alignment and repetition of of the arrows and bullet points. But for some purposes, the original design could be a little flat.

Adding color contrast makes the design pop, and it draws attention to important parts of the presentation and message.

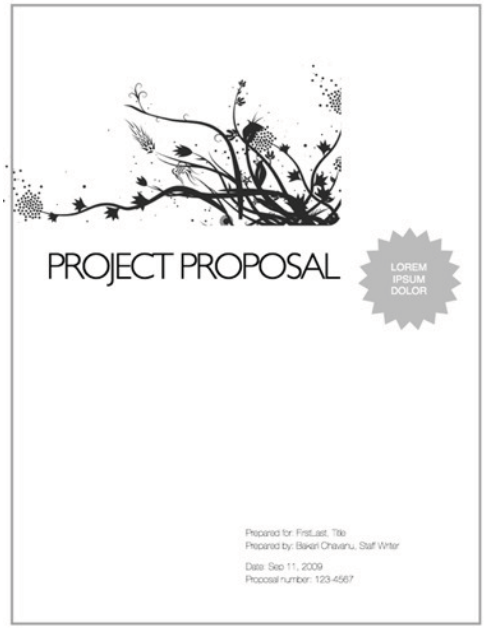
Notice another piece of contrast: the two arrows are followed by the check in the circle, which sends a visual message. The color of that element could also be changed to add contrast.

WHITE SPACE

Depending on the presentation, the use of white space can be very powerful in design. It's useful when you want to make a direct message, to stand out above the clutter

found in many graphic designs. In this Canva business card template, the empty space helps bring clarity to the message.

A card reader first sees the graphic element, then the owner's name, followed by the contact information. Elements on the card are balanced and uncluttered.



The same goes for this the coversheet of this Pages project proposal template. The white space provides room for the clean font style of the title, the graphic elements, and the grouped text. Don't be afraid of leaving white space in your design. As Robin Williams points out, white space can also be a form of contrast.

LEARNING GRAPHIC PRINCIPLES

I'm not a graphic designer, but years ago I learned a lot from working through the exercises of Robin Williams' book. Canva.com also provides several design tutorials that cover the above basic principles and several other design techniques. The site makes it easy for users to customize templates and save designs for later use.

Try your hand at applying the above principles to your next graphic design project, and let us know your thoughts, ideas, and tips for learning graphic design.

WHAT IS GRAPHIC DESIGN?

Graphic design is the art of visual communication through the use of images, words, and ideas to give information to the viewers. Graphic design can be used for advertising, or just for entertainment intended for the mind.

ALIGNMENT

Alignment in graphic design is the keeping of related objects in line.

BALANCE

Designs in balance (or equilibrium) have their parts arrangement planned, keeping a coherent visual pattern (color, shape, space). "Balance" is a concept based on human perception and the complex nature of the human senses of weight and proportion. Humans can evaluate these visual elements in several situations to find a sense of balance. A design composition does not have to be symmetrical or linear to be considered balanced, the balance is global to

all elements even the absence of content. In this context perfectly symmetrical and linear compositions are not necessarily balanced and so asymmetrical or radial distributions of text and graphic elements can achieve balance in a composition.

CONTRAST

Distinguishing by comparing/creating differences. Some ways of creating contrast among elements in the design include using contrasting colors, sizes, shapes, locations, or relationships. For text, contrast is achieved by mixing serif and sans-serif on the page, by using very different type styles, or by using type in surprising or unusual ways. Another way to describe contrast, is to say "a small object next to a large object will look smaller". As contrast in size diminishes, monotony is approached.

EMPHASIS

Making a specific element stand out or draw attention to the eye. Emphasis can be achieved in graphic design by placing elements on the page in positions where the eye is naturally drawn, by using other principles such as contrast, repetition, or movement. Bold and italic type provides emphasis for text. Graphic elements gain emphasis through size, visual weight, color, complexity, uniqueness, placement on the page, and other features.

GESTALT

Sometimes considered a distinct principle of design, gestalt is the concept that "the whole is greater than the

sum of its parts." Gestalt is a concept from psychology, where theorists note the propensity of humans to conceptually group things together to make a meaningful whole. When viewing designs, humans apply this principle unconsciously by seeing connections and relationships among and between the elements in the design. The overall perception of gestalt in a design is created through harmony, unity, balance, proportion, proximity, and other visual cues. Designers can use this principle to create visual connections and relationships that clarify and strengthen the overall "feel" and meaning of the design.

HARMONY

As with music, graphical elements can be said to be working in harmony - the individual parts come together as visually compelling and a meaningful whole. Disharmony can also be used just as it is in musical compositions: to enhance the emotional complexity, to challenge the viewer, and to give a contrast within the overall composition.

MOVEMENT

Movement is creating an instability, making motion to blur the image. Movement can be achieved by using graphic elements that direct the eye in a certain direction such as arrows that point the way overtly or a series of lines or dots that get progressively larger or smaller, creating a more subtle sense of movement. Movement can be accomplished simply by using a photograph or clip art of something moving - a runner - as opposed to something stationary - a person standing.

PROPORTION

This indicates the relative visual size and weight of particular graphical elements in a design composition.

PROXIMITY

Closeness or distance of individual design elements. Close proximity indicates a connection.

REPETITION

Repeating a sequence; having it occur more than a few times. In design, repetition creates visual consistency in page designs, such as using the same style of headlines, the same style of initial capitals, or repeating the same basic layout from one page to another.

Excessive repetition (monotony) may lead to boredom and uninteresting compositions. If one cannot avoid excessive repetitions for any reason, do not forget to add some visual breaks and white spaces where eyes can rest for a while.

RHYTHM

Successful designs have an effective ebb and flow. Text and Graphics should seem to be paced and patterned. Spacing is an effective application of this principle. Second, human beings are more comfortable with variation in general. Psychologically, most any serious lack in variation of anything (a solid, a line, a sound, a situation) can become very boring. Adding a little variation at non-specific intervals (every now and again) gives most any design an interesting appeal as long as it is not overdone. In setting type, rhythm can be created or disrupted. Compare the gibberish strings,

"as erav mono ewone zenao oro remuna oravanam" and "githol urtym reislyt quadirit". Notice how the latter seems to be more organic and readable than the former. This is resultant of two things. One, the eye more easily follows abnormalities and variation, like an ocular foothold. Too-narrow columns result in over-hyphenation. Images that interrupt a passage of text can break the rhythm for the reader and they could disturb the visual appearance of the page.

UNITY

Unity creates a feeling of wholeness. Unity is usually achieved when the parts complement each other in a way where they have something in common. Unity can be achieved by use of the same color, or different tints of it, or using a similar graphic style for illustrations.

WHITE SPACE

Areas of a design are devoid of text or graphics. White space includes margins, gutters, space between lines of type (leading), off-set of text from images (text wraps) and any other part of the page that is empty. White space is also analogous to "negative space" where "positive space" is defined as images, blocks of text, and other graphical elements. In graphic design, the white space, or negative space, is considered an important element of the overall design. It is used - and evaluated - based on the same criteria as the rest of the elements in the design. White space can add to or detract from the balance, unity, harmony, rhythm, and overall success of a design. White space can

give emphasis, contrast, and movement. It can be used for repetition and pattern, and work within various relationships with other elements of the positive and negative spaces in the design.

DESIGN ELEMENTS AND PRINCIPLES

Visual Design elements and principles describe fundamental ideas about the practice of good visual design.

As William Lidwell stated in *Universal Principles of Design*:

The best designers sometimes disregard the principles of design. When they do so, however, there is usually some compensating merit attained at the cost of the violation. Unless you are certain of doing as well, it is best to abide by the principles.

DESIGN ELEMENTS

Design elements are the basic units of a painting, drawing, design or other visual piece and include:

COLOR

- Color can play a large role in the elements of design with the color wheel being used as a tool, and color theory providing a body of practical guidance to color mixing and the visual impacts of specific color combination.

Uses:

- Color can aid organization so develop a color strategy and stay consistent with those colors.
- It can give emphasis to create a hierarchy to the piece of art

ATTRIBUTES

- Hue
- Values and tints and shades of colors that are created by adding black to a color for a shade and white for a tint. Creating a tint or shade of color reduces the saturation.
- Saturation gives a color brightness or dullness.

SHAPE

A shape is defined as a two or more dimensional area that stands out from the space next to or around it due to a defined or implied boundary, or because of differences of value, color, or texture. All objects are composed of shapes and all other 'Elements of Design' are shapes in some way.

CATEGORIES

- Mechanical Shapes or Geometric Shapes are the shapes that can be drawn using a ruler or compass. Mechanical shapes, whether simple or complex, produce a feeling of control or order.
- Organic Shapes are freehand drawn shapes that are complex and normally found in nature. Organic shapes produce a natural feel.

TEXTURE

Meaning the way a surface feels or is perceived to feel. Texture can be added to attract or repel interest to an element, depending on the pleasantness of the texture.

Types of texture:

- Tactile texture is the actual three-dimension feel of a surface that can be touched. Painter can use impasto to build peaks and create texture.
- Visual texture is the illusion of the surfaces peaks and valleys, like the tree pictured. Any texture shown in a photo is a visual texture, meaning the paper is smooth no matter how rough the image perceives it to be.

Most textures have a natural touch but still seem to repeat a motif in some way. Regularly repeating a motif will result in a texture appearing as a pattern.

SPACE

In design, space is concerned with the area deep within the moment of designated design, the design will take place on.

For a two-dimensional design, space concerns creating the illusion of a third dimension on a flat surface:

- Overlap is the effect where objects appear to be on top of each other. This illusion makes the top element look closer to the observer. There is no way to determine the depth of the space, only the order of closeness.
- Shading adds gradation marks to make an object of a two-dimensional surface seem three-dimensional.
- Highlight, Transitional Light, Core of the Shadow, Reflected Light, and Cast Shadow give an object a three-dimensional look.
- Linear Perspective is the concept relating to how an object seems smaller the farther away it gets.

- Atmospheric Perspective is based on how air acts as a filter to change the appearance of distance objects.

FORM

Form may be described as any three-dimensional object. Form can be measured, from top to bottom (height), side to side (width), and from back to front (depth). Form is also defined by light and dark. It can be defined by the presence of shadows on surfaces or faces of an object. There are two types of form, geometric (man-made) and natural (organic form). Form may be created by the combining of two or more shapes. It may be enhanced by tone, texture and color. It can be illustrated or constructed.

COMPUTER AIDED INDUSTRIAL DESIGN

Computer-aided industrial design (CAID) is a subset of computer-aided design (CAD) that includes software that directly helps in product development. Within CAID programmes designers have the freedom of creativity, but typically follow a simple design methodology:

- Creating sketches, using a stylus
- Generating curves directly from the sketch
- Generating surfaces directly from the curves

The end result is a 3D model that projects the main design intent the designer had in mind. The model can then be saved in STL format to send it to a rapid prototyping machine to create the real-life model. CAID helps the designer to focus on the technical part of the design methodology rather than taking care of sketching and modeling—then

contributing to the selection of a better product proposal in less time.

Later, when the requisites and parameters of the product have been defined by means of using CAID software, the designer can import the result of his work into a CAD programme (typically a Solid Modeler) for adjustments prior to production and generation of blueprints and manufacturing processes. What differentiates CAID from CAD is that the former is far more conceptual and less technical than the latter. Within a CAID programme, the designer can express him/herself without extents, whilst in CAD software there is always the manufacturing factor.

2

Virtual Design and Construction

Virtual Design to Construction (VDC) is the management of integrated multi-disciplinary performance models of design-construction projects, including the Product (i.e., facilities), Work Processes and Organization of the design - construction - operation team in order to support explicit and public business objectives.

The theoretical basis of VDC includes:

- Engineering modeling methods: product, organization, process
- Analysis methods (model-based): including schedule, cost, 4D interactions and process risks, these are termed BIM tools
- Visualization methods
- Business metrics and focus on strategic management
- Economic Impact analysis (i.e., models of both the cost and value of capital investments)

VDC PROJECT MANAGER

"The production of a Building Information Model (BIM) for the construction of a project involves the use of an integrated multi-disciplinary performance model to encompass the building geometry, spatial relationships, geographic information, along with quantities and properties of the building components. The Virtual Design to Construction Project Manager (VDC - also known as VDCPM) is a professional in the field of project management and delivery.

The VDC is retained by a design build team on the clients' behalf from the pre-design phase through certificate of occupancy in order to develop and to track the object oriented BIM against predicted and measured performance objectives. The VDC manages the project delivery through multi-disciplinary building information models that drive analysis, schedules, take-off, and logistics. The VDC is skilled in the use of BIM as a tool to manage and assess the technology, staff, and procedural needs of a project. In short the VDC is a contemporary project managing architect who is equipped to deal with the current evolution of project delivery. The VDC acts as a conduit to bridge time tested construction knowledge to digital analysis and representation. VDC position avoids the well intentioned failures created by competent managers who lack the knowledge to implement the technology for which they are entrusted. Recent economic conditions have placed a spot light on industry wide deficiency in the organization of architectural staff, the lack of interoperability of project generated information, and the amount of non-beneficial

redundancy which eventually finds its way to the client through an inferior project with increased cost. The VDC fulfills a critical role in contemporary project delivery in part due to the single platform integration of sketch tools, massing, solid modeling, analysis, & rendering organized within a singular object change engine. Available technology removes the need for digital redundancies and file conversions at each stage of design. Information can be tracked and managed from inception to project delivery with the use of a qualified VDC who secures the clients return on investment by tracking stated project performance objectives. The development of virtual design tools from 1957 to 2007 has created a digital landfill of applications, many whose continued use has hindered progress all the while accelerating Architect, Engineering, Contractor costs without increased accuracy, efficiency, or integration of disciplines."

VDC MANAGED BIM PROJECT MODEL

"Virtual Design to Construction BIM models are virtual because they show computer-based descriptions of the project. The BIM project model emphasizes those aspects of the project that can be designed and managed, i.e., the *product* (typically a building or plant), the *organization* that will define, design, construct and operate it, and the *process* that the organization teams will follow, or POP. These models are logically integrated in the sense that they all can access shared data, and if a user highlights or changes an aspect of one, the integrated models can highlight or change the dependent aspects of related models. The models are multi-

disciplinary in the sense that they represent the Architect, Engineering, contractor (AEC) and Owner of the project, as well as relevant sub disciplines. The models are performance models in the sense that they predict some aspects of project performance, track many that are relevant, and can show predicted and measured performance in relationship to stated project performance objectives. Some companies now practice the first steps of BIM modeling, and they consistently find that they improve business performance by doing so."

CONSTRUCTION INDUSTRY BIM TOOLS AND METHODOLOGIES UTILIZED BY VDC

BIM SOFTWARE TOOLS

- ArchiCAD from Graphisoft
- Building Explorer
- Autodesk Navisworks JetStream 4D
- Autodesk Revit
- Autodesk AutoCAD Civil 3D
- Tekla Structures from Tekla Corporation
- Advance Concrete
- Advance Steel
- Microstation

VDC RELATED METHODOLOGIES

- Semantic integration
- Learning-by-doing
- Deductive-nomological model

- Scientific evidence
- Hypothesis
- Qualitative research
- Quantitative research
- Case-based reasoning
- Action research
- Power of a method
- Upper ontology within the Ontology domain
- Schema representation
- Work breakdown structure
- Object-oriented programming

3D FLOOR PLAN

A 3D floor plan, or 3D floorplan, is a virtual model of a building floor plan, depicted from a birds eye view, utilized within the building industry to better convey archtitectural plans. Usually built to scale, a 3D floor plan must include walls and a floor and typically includes exterior wall fenestrations, windows, and doorways. It does not include a ceiling so as not to obstruct the view. Other common attributes may be added, but are not required, such as cabinets, flooring, bathroom fixtures, paint color, wall tile, and other interior finishes. Furniture may be added to assist in communicating proper home staging and interior design.

PURPOSE

3D floor plans assist real estate marketers and architects in explaining floor plans to clients. Their simplicity allows individuals unfamiliar with conventional floor plans to

understand difficult architectural concepts. This allows architects and homeowners to literally see design elements prior to construction and alter design elements during the design phase. 3D floorplans are often commissioned by architects, builders, hotels, universities, real estate agents, and property owners to assist in relating their floor plans to clients.

CONSTRUCTION

A 3d floor plan is built utilizing advanced 3d rendering software, the same type of software used to create major animated motion pictures. Through complex lighting, staging, camera, and rendering techniques 3D floorplans appear to be real photographs rather than digital representations of the buildings they are modeled after.

3

Electronic Design Automation

Electronic design automation (EDA or ECAD) is a category of software tools for designing electronic systems such as printed circuit boards and integrated circuits. The tools work together in a design flow that chip designers use to design and analyze entire semiconductor chips.

HISTORY

EARLY DAYS

Before EDA, integrated circuits were designed by hand, and manually laid out. Some advanced shops used geometric software to generate the tapes for the Gerber photoplotter, but even those copied digital recordings of mechanically-drawn components. The process was fundamentally graphic, with the translation from electronics to graphics done manually. The best known company from this era was Calma, whose GDSII format survives. By the mid-70s,

developers started to automate the design, and not just the drafting. The first placement and routing (Place and route) tools were developed. The proceedings of the Design Automation Conference cover much of this era. The next era began about the time of the publication of "Introduction to VLSI Systems" by Carver Mead and Lynn Conway in 1980. This ground breaking text advocated chip design with programming languages that compiled to silicon.

The immediate result was a considerable increase in the complexity of the chips that could be designed, with improved access to design verification tools that used logic simulation. Often the chips were easier to lay out and more likely to function correctly, since their designs could be simulated more thoroughly prior to construction. Although the languages and tools have evolved, this general approach of specifying the desired behavior in a textual programming language and letting the tools derive the detailed physical design remains the basis of digital IC design today. The earliest EDA tools were produced academically. One of the most famous was the "Berkeley VLSI Tools Tarball", a set of UNIX utilities used to design early VLSI systems. Still widely used is the Espresso heuristic logic minimizer and Magic. Another crucial development was the formation of MOSIS, a consortium of universities and fabricators that developed an inexpensive way to train student chip designers by producing real integrated circuits. The basic concept was to use reliable, low-cost, relatively low-technology IC processes, and pack a large number of projects per wafer, with just a few copies of each projects' chips. Cooperating fabricators either donated the processed

wafers, or sold them at cost, seeing the programme as helpful to their own long-term growth.

BIRTH OF COMMERCIAL EDA

1981 marks the beginning of EDA as an industry. For many years, the larger electronic companies, such as Hewlett Packard, Tektronix, and Intel, had pursued EDA internally. In 1981, managers and developers spun out of these companies to concentrate on EDA as a business. Daisy Systems, Mentor Graphics, and Valid Logic Systems were all founded around this time, and collectively referred to as DMV. Within a few years there were many companies specializing in EDA, each with a slightly different emphasis. The first trade show for EDA was held at the Design Automation Conference in 1984. In 1986, Verilog, a popular high-level design language, was first introduced as a hardware description language by Gateway Design Automation. In 1987, the U.S. Department of Defense funded creation of VHDL as a specification language. Simulators quickly followed these introductions, permitting direct simulation of chip designs: executable specifications. In a few more years, back-ends were developed to perform logic synthesis.

CURRENT STATUS

Current digital flows are extremely modular (see Integrated circuit design, Design closure, and Design flow (EDA)). The front ends produce standardized design descriptions that compile into invocations of "cells,", without regard to the cell technology. Cells implement logic or other electronic functions

using a particular integrated circuit technology. Fabricators generally provide libraries of components for their production processes, with simulation models that fit standard simulation tools. Analog EDA tools are far less modular, since many more functions are required, they interact more strongly, and the components are (in general) less ideal. EDA for electronics has rapidly increased in importance with the continuous scaling of semiconductor technology. Some users are foundry operators, who operate the semiconductor fabrication facilities, or "fabs", and design-service companies who use EDA software to evaluate an incoming design for manufacturing readiness. EDA tools are also used for programming design functionality into FPGAs.

SOFTWARE FOCUSES

DESIGN

- High-level synthesis(syn. behavioural synthesis, algorithmic synthesis) For digital chips
- Logic synthesis translation of abstract, logical language such as Verilog or VHDL into a discrete netlist of logic-gates
- Schematic Capture For standard cell digital, analog, rf like Capture CIS in Orcad by CADENCE and ISIS in Proteus
- Layout like Layout in Orcad by Cadence, ARES in Proteus

DESIGN FLOWS

Design flows are the explicit combination of electronic design automation tools to accomplish the design of an

integrated circuit. Moore's law has driven the entire IC implementation RTL to GDSII design flows from one which uses primarily standalone synthesis, placement, and routing algorithms to an integrated construction and analysis flows for design closure. The challenges of rising interconnect delay led to a new way of thinking about and integrating design closure tools. New scaling challenges such as leakage power, variability, and reliability will keep on challenging the current state of the art in design closure. The RTL to GDSII flow underwent significant changes from 1980 through 2005. The continued scaling of CMOS technologies significantly changed the objectives of the various design steps.

The lack of good predictors for delay has led to significant changes in recent design flows. Challenges like leakage power, variability, and reliability will continue to require significant changes to the design closure process in the future. Many factors describe what drove the design flow from a set of separate design steps to a fully integrated approach, and what further changes are coming to address the latest challenges. In his keynote at the 40th Design Automation Conference entitled The Tides of EDA, Alberto Sangiovanni-Vincentelli distinguished three periods of EDA: The Age of the Gods, The Age of the Heroes, and The Age of the Men. These eras were characterized respectively by senses, imagination, and reason. When we limit ourselves to the RTL to GDSII flow of the CAD area, we can distinguish three main eras in its development: the Age of Invention, the Age of Implementation, and the Age of Integration.

- The Age of Invention: During the invention era, routing, placement, static timing analysis and logic synthesis were invented.
- The Age of Implementation: In the age of implementation, these steps were drastically improved by designing sophisticated data structures and advanced algorithms. This allowed the tools in each of these design steps to keep pace with the rapidly increasing design sizes. However, due to the lack of good predictive cost functions, it became impossible to execute a design flow by a set of discrete steps, no matter how efficiently each of the steps was implemented.
- The Age of Integration: This led to the age of integration where most of the design steps are performed in an integrated environment, driven by a set of incremental cost analyzers.

SIMULATION

- Transistor simulation – low-level transistor-simulation of a schematic/layout's behavior, accurate at device-level.
- Logic simulation – digital-simulation of an RTL or gate-netlist's digital (boolean 0/1) behavior, accurate at boolean-level.
- Behavioral Simulation – high-level simulation of a design's architectural operation, accurate at cycle-level or interface-level.
- Hardware emulation – Use of special purpose hardware to emulate the logic of a proposed design.

Can sometimes be plugged into a system in place of a yet-to-be-built chip; this is called in-circuit emulation.

- Technology CAD simulate and analyze the underlying process technology. Electrical properties of devices are derived directly from device physics.
- Electromagnetic field solvers, or just field solvers, solve Maxwell's equations directly for cases of interest in IC and PCB design. They are known for being slower but more accurate than the layout extraction above.

ELECTRONIC CIRCUIT SIMULATION

Electronic circuit simulation uses mathematical models to replicate the behavior of an actual electronic device or circuit. Simulation software allows for modeling of circuit operation and is an invaluable analysis tool. Due to its highly accurate modeling capability, many Colleges and Universities use this type of software for the teaching of electronics technician and electronics engineering programmes.

Electronics simulation software engages the user by integrating them into the learning experience. These kinds of interactions actively engage learners to analyze, synthesize, organize, and evaluate content and result in learners constructing their own knowledge. Simulating a circuit's behavior before actually building it can greatly improve design efficiency by making faulty designs known as such, and providing insight into the behavior of electronics circuit designs.

In particular, for integrated circuits, the tooling (photomasks) is expensive, breadboards are impractical, and probing the behavior of internal signals is extremely difficult. Therefore almost all IC design relies heavily on simulation. The most well known analog simulator is SPICE. Probably the best known digital simulators are those based on Verilog and VHDL. Some electronics simulators integrate a schematic editor, a simulation engine, and on-screen waveforms, and make "what-if" scenarios easy and instant. They also typically contain extensive model and device libraries. These models typically include IC specific transistor models such as BSIM, generic components such as resistors, capacitors, inductors and transformers, user defined models (such as controlled current and voltage sources, or models in Verilog-A or VHDL-AMS). Printed circuit board (PCB) design requires specific models as well, such as transmission lines for the traces and IBIS models for driving and receiving electronics.

TYPES

While there are strictly analog electronics circuit simulators, popular simulators often include both analog and event-driven digital simulation capabilities, and are known as mixed-mode simulators. This means that any simulation may contain components that are analog, event driven (digital or sampled-data), or a combination of both. An entire mixed signal analysis can be driven from one integrated schematic. All the digital models in mixed-mode simulators provide accurate specification of propagation time and rise/fall time delays.

The event driven algorithm provided by mixed-mode simulators is general purpose and supports non-digital types of data. For example, elements can use real or integer values to simulate DSP functions or sampled data filters. Because the event driven algorithm is faster than the standard SPICE matrix solution, simulation time is greatly reduced for circuits that use event driven models in place of analog models. Mixed-mode simulation is handled on three levels; (a) with primitive digital elements that use timing models and the built-in 12 or 16 state digital logic simulator, (b) with subcircuit models that use the actual transistor topology of the integrated circuit, and finally, (c) with In-line Boolean logic expressions.

Exact representations are used mainly in the analysis of transmission line and signal integrity problems where a close inspection of an IC's I/O characteristics is needed. Boolean logic expressions are delay-less functions that are used to provide efficient logic signal processing in an analog environment. These two modeling techniques use SPICE to solve a problem while the third method, digital primitives, use mixed mode capability. Each of these methods has its merits and target applications. In fact, many simulations (particularly those which use A/D technology) call for the combination of all three approaches. No one approach alone is sufficient. Another type of simulation used mainly for power electronics represent piecewise linear algorithms. These algorithms use an analog (linear) simulation until a power electronic switch changes its state. At this time a new analog model is calculated to be used for the next simulation period. This methodology both enhances simulation speed and stability significantly.

COMPLEXITIES

Often circuit simulators do not take into account the process variations that occur when the design is fabricated into silicon. These variations can be small, but taken together can change the output of a chip significantly. Process variations occur in the manufacture of circuits in silicon. Temperature variation can also be modeled to simulate the circuit's performance through temperature ranges.

ANALYSIS AND VERIFICATION

- Functional verification
- Clock Domain Crossing Verification (CDC check): Similar to linting, but these checks/tools specialize in detecting and reporting potential issues like data loss, meta-stability due to use of multiple clock domains in the design.
- Formal verification, also model checking: Attempts to prove, by mathematical methods, that the system has certain desired properties, and that certain undesired effects (such as deadlock) cannot occur.
- Equivalence checking: algorithmic comparison between a chip's RTL-description and synthesized gate-netlist, to ensure functional equivalence at the *logical* level.
- Static timing analysis: Analysis of the timing of a circuit in an input-independent manner, hence finding a worst case over all possible inputs.
- Physical verification, PV: checking if a design is physically manufacturable, and that the resulting chips will not have any function-preventing physical defects, and will meet original specifications.

MANUFACTURING PREPARATION

- Mask data preparation, MDP: generation of actual lithography photomask used to physically manufacture the chip.
  - o Resolution enhancement techniques, RET – methods of increasing of quality of final photomask.
  - o Optical proximity correction, OPC – up-front compensation for diffraction and interference effects occurring later when chip is manufactured using this mask.
  - o Mask generation – generation of flat mask image from hierarchical design.
  - o Automatic test pattern generation, ATPG – generates pattern-data to systematically exercise as many logic-gates, and other components, as possible.
  - o Built-in self-test, or BIST – installs self-contained test-controllers to automatically test a logic (or memory) structure in the design

COMPANIES

For more details on this topic, see List of EDA companies.

TOP COMPANIES

- $3.73 billion - Synopsys
- $2.06 billion - Cadence
- $1.18 billion - Mentor Graphics
- $233 million - Magma Design Automation
- $157 million - Zuken Inc.

Note: Market caps current as of October, 2010. EEsof should likely be on this list, but does not have a market cap as it is the EDA division of Agilent.

ACQUISITIONS

Many of the EDA companies acquire small companies with software or other technology that can be adapted to their core business. Most of the market leaders are rather incestuous amalgamations of many smaller companies. This trend is helped by the tendency of software companies to design tools as accessories that fit naturally into a larger vendor's suite of programmes (on digital circuitry, many new tools incorporate analog design, and mixed systems. This is happening because there is now a trend to place entire electronic systems on a single chip.

COMPUTER GRAPHICS

The development of computer graphics has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry. The term computer graphics has been used in a broad sense to describe "almost everything on computers that is not text or sound". Typically, the term *computer graphics* refers to several different things:

- the representation and manipulation of image data by a computer
- the various technologies used to create and manipulate images
- the images so produced, and
- the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Today, computers and computer-generated images touch many aspects of daily life. Computer imagery is found on television, in newspapers, for example in weather reports, or for example in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media "such graphs are used to illustrate papers, reports, thesis", and other presentation material. Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used.

Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component". The advance in computer graphics was to come from Ivan Sutherland. In 1961 Sutherland created another computer drawing programme called Sketchpad. Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted

an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location. Sutherland seemed to find the perfect solution for many of the graphics problems he faced.

Even today, many standards of computer graphics interfaces got their start with this early Sketchpad programme. One example of this is in drawing constraints. If one wants to draw a square for example, s/he doesn't have to worry about drawing four lines perfectly to form the edges of the box. One can simply specify that s/he wants to draw a box, and then specify the location and size of the box. The software will then construct a perfect box, with the right dimensions and at the right location. Another example is that Sutherland's software modeled objects - not just a picture of objects. In other words, with a model of a car, one could change the size of the tires without affecting the rest of the car. It could stretch the body of the car without deforming the tires. These early computer graphics were Vector graphics, composed of thin lines whereas modern day graphics are Raster based using pixels. The difference between vector graphics and raster graphics can be illustrated with a shipwrecked sailor.

He creates an SOS sign in the sand by arranging rocks in the shape of the letters "SOS." He also has some brightly colored rope, with which he makes a second "SOS" sign by arranging the rope in the shapes of the letters. The rock

SOS sign is similar to raster graphics. Every pixel has to be individually accounted for. The rope SOS sign is equivalent to vector graphics. The computer simply sets the starting point and ending point for the line and perhaps bend it a little between the two end points. The disadvantages to vector files are that they cannot represent continuous tone images and they are limited in the number of colors available. Raster formats on the other hand work well for continuous tone images and can reproduce as many colors as needed. Also in 1961 another student at MIT, Steve Russell, created the first video game, Spacewar. Written for the DEC PDP-1, Spacewar was an instant success and copies started flowing to other PDP-1 owners and eventually even DEC got a copy. The engineers at DEC used it as a diagnostic programme on every new PDP-1 before shipping it. The sales force picked up on this quickly enough and when installing new units, would run the world's first video game for their new customers.

E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963. In this computer generated film, Zajac showed how the attitude of a satellite could be altered as it orbits the Earth. He created the animation on an IBM 7090 mainframe computer. Also at BTL, Ken Knowlton, Frank Sindon and Michael Noll started working in the computer graphics field. Sindon created a film called Force, Mass and Motion illustrating Newton's laws of motion in operation.

Around the same time, other scientists were creating computer graphics to illustrate their research. At Lawrence

Radiation Laboratory, Nelson Max created the films, "Flow of a Viscous Fluid" and "Propagation of Shock Waves in a Solid Form." Boeing Aircraft created a film called "Vibration of an Aircraft." It wasn't long before major corporations started taking an interest in computer graphics. TRW, Lockheed-Georgia, General Electric and Sperry Rand are among the many companies that were getting started in computer graphics by the mid 1960's. IBM was quick to respond to this interest by releasing the IBM 2250 graphics terminal, the first commercially available graphics computer. Ralph Baer, a supervising engineer at Sanders Associates, came up with a home video game in 1966 that was later licensed to Magnavox and called the Odyssey. While very simplistic, and requiring fairly inexpensive electronic parts, it allowed the player to move points of light around on a screen. It was the first consumer computer graphics product.

Also in 1966, Sutherland at MIT invented the first computer controlled head-mounted display (HMD). Called the Sword of Damocles because of the hardware required for support, it displayed two separate wireframe images, one for each eye. This allowed the viewer to see the computer scene in stereoscopic 3D. After receiving his Ph.D. from MIT, Sutherland became Director of Information Processing at ARPA (Advanced Research Projects Agency), and later became a professor at Harvard. Dave Evans was director of engineering at Bendix Corporation's computer division from 1953 to 1962, after which he worked for the next five years as a visiting professor at Berkeley. There he continued his interest in computers and how they interfaced with people. In 1968 the University of Utah recruited Evans to

form a computer science programme, and computer graphics quickly became his primary interest. This new department would become the world's primary research center for computer graphics. In 1967 Sutherland was recruited by Evans to join the computer science programme at the University of Utah. There he perfected his HMD. Twenty years later, NASA would re-discover his techniques in their virtual reality research.

At Utah, Sutherland and Evans were highly sought after consultants by large companies but they were frustrated at the lack of graphics hardware available at the time so they started formulating a plan to start their own company. A student by the name of Edwin Catmull started at the University of Utah in 1970 and signed up for Sutherland's computer graphics class. Catmull had just come from The Boeing Company and had been working on his degree in physics. Growing up on Disney, Catmull loved animation yet quickly discovered that he didn't have the talent for drawing. Now Catmull (along with many others) saw computers as the natural progression of animation and they wanted to be part of the revolution. The first animation that Catmull saw was his own. He created an animation of his hand opening and closing. It became one of his goals to produce a feature length motion picture using computer graphics. In the same class, Fred Parke created an animation of his wife's face.

Because of Evan's and Sutherland's presence, UU was gaining quite a reputation as the place to be for computer graphics research so Catmull went there to learn 3D animation. As the UU computer graphics laboratory was

attracting people from all over, John Warnock was one of those early pioneers; he would later found Adobe Systems and create a revolution in the publishing world with his PostScript page description language. Tom Stockham led the image processing group at UU which worked closely with the computer graphics lab. Jim Clark was also there; he would later found Silicon Graphics, Inc. The first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

IMAGE TYPES

2D COMPUTER GRAPHICS

2D computer graphics are the computer-based generation of digital images—mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them. 2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, advertising, etc.. In those applications, the two-dimensional image is not just a representation of a real-world object, but an independent artifact with added semantic value; two-dimensional models are therefore preferred, because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

PIXEL ART

Pixel art is a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level. Graphics in most old (or relatively limited) computer and video games, graphing calculator games, and many mobile phone games are mostly pixel art.

VECTOR GRAPHICS

Vector graphics formats are complementary to raster graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images Vector graphics consists in encoding information about shapes and colors that comprise the image, which can allow for more flexibility in rendering. There are instances when working with vector tools and formats is best practice, and instances when working with raster tools and formats is best practice. There are times when both formats come together. An understanding of the advantages and limitations of each technology and the relationship between them is most likely to result in efficient and effective use of tools.

3D COMPUTER GRAPHICS

3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. Such images may be for later display or for real-time viewing. Despite these differences, 3D computer graphics rely on many of the same algorithms as

2D computer vector graphics in the wire frame model and 2D computer raster graphics in the final rendered display. In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and primarily 3D may use 2D rendering techniques. 3D computer graphics are often referred to as 3D models. Apart from the rendered graphic, the model is contained within the graphical data file. However, there are differences. A 3D model is the mathematical representation of any three-dimensional object. A model is not technically a graphic until it is visually displayed. Due to 3D printing, 3D models are not confined to virtual space. A model can be displayed visually as a two-dimensional image through a process called *3D rendering*, or used in non-graphical computer simulations and calculations. There are some 3D computer graphics software for users to create 3D images.

COMPUTER ANIMATION

Computer animation is the art of creating moving images via the use of computers. It is a subfield of computer graphics and animation. Increasingly it is created by means of 3D computer graphics, though 2D computer graphics are still widely used for stylistic, low bandwidth, and faster real-time rendering needs. Sometimes the target of the animation is the computer itself, but sometimes the target is another medium, such as film. It is also referred to as CGI (Computer-generated imagery or computer-generated imaging), especially when used in films. Virtual entities may contain and be controlled by assorted attributes, such as transform

values (location, orientation, and scale) stored in an object's transformation matrix. Animation is the change of an attribute over time. Multiple methods of achieving animation exist; the rudimentary form is based on the creation and editing of keyframes, each storing a value at a given time, per attribute to be animated. The 2D/3D graphics software will interpolate between keyframes, creating an editable curve of a value mapped over time, resulting in animation.

Other methods of animation include procedural and expression-based techniques: the former consolidates related elements of animated entities into sets of attributes, useful for creating particle effects and crowd simulations; the latter allows an evaluated result returned from a user-defined logical expression, coupled with mathematics, to automate animation in a predictable way (convenient for controlling bone behavior beyond what a hierarchy offers in skeletal system set up). To create the illusion of movement, an image is displayed on the computer screen then quickly replaced by a new image that is similar to the previous image, but shifted slightly. This technique is identical to the illusion of movement in television and motion pictures.

CONCEPTS AND PRINCIPLES

Images are typically produced by optical devices;such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces. A digital image is a representation of a two-dimensional image in binary format as a sequence of ones and zeros. Digital images include both vector images and raster images, but raster images are more commonly used.

PIXEL

In digital imaging, a pixel (or picture element) is a single point in a raster image. Pixels are normally arranged in a regular 2-dimensional grid, and are often represented using dots or squares. Each pixel is a sample of an original image, where more samples typically provide a more accurate representation of the original. The intensity of each pixel is variable; in color systems, each pixel has typically three components such as red, green, and blue.

GRAPHICS

Graphics are visual presentations on some surface, such as a wall, canvas, computer screen, paper, or stone to brand, inform, illustrate, or entertain. Examples are photographs, drawings, line art, graphs, diagrams, typography, numbers, symbols, geometric designs, maps, engineering drawings, or other images. Graphics often combine text, illustration, and color. Graphic design may consist of the deliberate selection, creation, or arrangement of typography alone, as in a brochure, flier, poster, web site, or book without any other element. Clarity or effective communication may be the objective, association with other cultural elements may be sought, or merely, the creation of a distinctive style.

RENDERING

Rendering is the process of generating an image from a model (or models in what collectively could be called a *scene* file), by means of computer programmes. A scene file contains objects in a strictly defined language or data structure; it

would contain geometry, viewpoint, texture, lighting, and shading information as a description of the virtual scene. The data contained in the scene file is then passed to a rendering programme to be processed and output to a digital image or raster graphics image file. The rendering programme is usually built into the computer graphics software, though others are available as plug-ins or entirely separate programmes. The term "rendering" may be by analogy with an "artist's rendering" of a scene. Though the technical details of rendering methods vary, the general challenges to overcome in producing a 2D image from a 3D representation stored in a scene file are outlined as the graphics pipeline along a rendering device, such as a GPU. A GPU is a purpose-built device able to assist a CPU in performing complex rendering calculations. If a scene is to look relatively realistic and predictable under virtual lighting, the rendering software should solve the rendering equation. The rendering equation doesn't account for all lighting phenomena, but is a general lighting model for computer-generated imagery. 'Rendering' is also used to describe the process of calculating effects in a video editing file to produce final video output.

3D PROJECTION

3D projection is a method of mapping three dimensional points to a two dimensional plane. As most current methods for displaying graphical data are based on planar two dimensional media, the use of this type of projection is widespread, especially in computer graphics, engineering and drafting.

RAY TRACING

Ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane. The technique is capable of producing a very high degree of photorealism; usually higher than that of typical scanline rendering methods, but at a greater computational cost.

SHADING

Shading refers to depicting depth in 3D models or illustrations by varying levels of darkness. It is a process used in drawing for depicting levels of darkness on paper by applying media more densely or with a darker shade for darker areas, and less densely or with a lighter shade for lighter areas. There are various techniques of shading including cross hatching where perpendicular lines of varying closeness are drawn in a grid pattern to shade an area. The closer the lines are together, the darker the area appears. Likewise, the farther apart the lines are, the lighter the area appears. The term has been recently generalized to mean that shaders are applied.

TEXTURE MAPPING

Texture mapping is a method for adding detail, surface texture, or colour to a computer-generated graphic or 3D model. Its application to 3D graphics was pioneered by Dr Edwin Catmull in 1974. A texture map is applied (mapped) to the surface of a shape, or polygon. This process is akin to applying patterned paper to a plain white box. Multitexturing is the use of more than one texture at a time on a polygon. Procedural textures (created from adjusting

parameters of an underlying algorithm that produces an output texture), and bitmap textures (created in an image editing application) are, generally speaking, common methods of implementing texture definition from a 3D animation programme, while intended placement of textures onto a model's surface often requires a technique known as UV mapping.

ANTI-ALIASING

Rendering resolution-independent entities (such as 3D models) for viewing on a raster (pixel-based) device such as a LCD display or CRT television inevitably causes aliasing artifacts mostly along geometric edges and the boundaries of texture details; these artifacts are informally called "jaggies". Anti-aliasing methods rectify such problems, resulting in imagery more pleasing to the viewer, but can be somewhat computationally expensive. Various anti-aliasing algorithms (such as supersampling) are able to be employed, then customized for the most efficient rendering performance versus quality of the resultant imagery; a graphics artist should consider this trade-off if anti-aliasing methods are to be used. A pre-anti-aliased bitmap texture being displayed on a screen (or screen location) at a resolution different than the resolution of the texture itself (such as a textured model in the distance from the virtual camera) will exhibit aliasing artifacts, while any procedurally-defined texture will always show aliasing artifacts as they are resolution-independent; techniques such as mipmapping and texture filtering help to solve texture-related aliasing problems.

4

The Evolution of Web Design

Since the first websites in the early 1992 s, designers have been experimenting with the way websites look. Early sites were entirely text-based, with minimal images and no real layout to speak of other than headings and paragraphs. However, the industry progressed, eventually bringing us table-based designs, then Flash, and finally CSS-based designs.

This article covers the brief history of the different eras of web design, including a handful of examples of each type of design.

THE FIRST WEB PAGES

In August 1991, Tim Berners-Lee published the first website, a simple, text-based page with a few links. A copy from 1992 of the original page still exists online. It had a dozen or so links, and simply served to tell people what the

World Wide Web was all about. Subsequent pages were similar, in that they were entirely text-based and had a single-column design with inline links. Initial versions of HTML (HyperText Markup Language) only allowed for very basic content structure: headings (<h1>, <h2>, etc.), paragraphs (<p>), and links (<a>). Subsequent versions of HTML allowed the addition of images (<img>) to pages, and eventually support for tables (<table>) was added.

WORLD WIDE WEB CONSORTIUM IS FORMED

In 1994, the World Wide Web Consortium (W3C) was established, and they set HTML as the standard for marking up web pages. This discouraged any single company from building a proprietary browser and programming language, which could have had a detrimental effect on the web as a whole. The W3C continues to set standards for open web markup and programming languages (such as JavaScript).

DESIGN ELEMENTS AND PRINCIPLES

Design elements and principles describe fundamental ideas about the practice of good visual design.

As William Lidwell's stated in *Universal Principles of Design*:

"The best designers sometimes disregard the principles of design. When they do so, however, there is usually some compensating merit attained at the cost of the violation. Unless you are certain of doing as well, it is best to abide by the principles."

These principles, which may overlap, are used in all visual design fields, including graphic design, industrial design, architecture and fine art.

DESIGN ELEMENTS

Design elements are the basic units of a painting, drawing, design or other visual piece and include:

LINE

A fundamental mark or stroke used in drawing in which the length is longer than the width. It is straight and has two connected points form a line and every line has a length, width, and direction it is straight.

Uses for lines in design

- Contour line: A line that defines or bounds an edge, but not always the outside edge, could represent a fold or color change.
- Divide space: A line that defines the edge of space can also be created by a gap of negative space. Many uses include to separate columns, rows of type, or to show a change in document type.
- Decoration: Lines are used in linear shapes and patterns to decorate many different substrates, and can be used to create shadows representing tonal value, called hatching.

COLOR

Color can play a large role in the elements of design with the color wheel being used as a tool, and color theory providing a body of practical guidance to color mixing and the visual impacts of specific color combination.

Types of color

- Primary color: The three colors that are equal distant on the color wheel and used to make up all other colors; red, yellow, and blue.

- Secondary color: A mixture of two primary colors including green, violet, and orange. Secondary colors are a way to have more vibrant colors.
- Tertiary color: Colors formed from a primary and a secondary color like yellow-green, red-violet, and yellow-orange.

Perceptual attributes of color

- Hue: The redness, blueness, and greenness of a color.
- Value (lightness): Tints and shades of colors that are created by adding black to a color for a shade and white for a tint. Creating a tint or shade of a color reduces the saturation.
- Saturation: Give a color brightness or dullness.

Ways color can guide the reader

- Aids organization: Develop a color strategy and stay consistent with those colors.
- Gives emphasis: Create a hierarchy
- Provides direction: Using warm and cool colors to relate parts with each other. Warm colors move elements forward and cool colors move them back. Display text using warm colors behind a cool color background will stand out and direct the readers eye.

SHAPE

A shape is defined as an area that stands out from the space next to or around it due to a defined or implied boundary, or because of differences of value, color, or texture. All objects are composed of shapes and all other 'Elements of Design' are shapes in some way.

General Categories of Shapes

- Mechanical Shapes (Geometric Shapes): These are the shapes that can be drawn using a ruler or compass. Mechanical shapes, whether simple or complex, produce a feeling of control or order.
- Organic Shapes: Freehand drawn shapes that are complex and normally found in nature. Organic shapes produce a natural feel.

TEXTURE (VISUAL ARTS)

Meaning the way a surface feels or is perceived to feel. Texture can be added to attract or repel interest to an element, depending on the pleasantness of the texture.

Types of texture

- Tactile texture: The actual three-dimension feel of a surface that can be touched. Painter can use impasto to build peaks and create texture.
- Visual texture: The illusion of the surfaces peaks and valleys, like the tree pictured. Any texture shown in a photo is a visual texture, meaning the paper is smooth no matter how rough the image perceives it to be.

Most textures have a natural feel but still seem to repeat a motif in some way. Regularly repeating a motif will result in a texture appearing as apattern.

SPACE

In design, space is concerned with the area the design will take place on. For a two-dimensional design space concerns creating the illusion of a third dimension on a flat surface.

Major Methods of Controlling the Illusion of Space

- Overlap: Where objects appear to be on top of each other. This illusion makes the top element look closer to the observer. There is no way to determine the depth of the space, only the order of closeness.
- Shading: Adding gradation marks to make an object of a two-dimensional surface seem three-dimensional.
- Five Kinds of Shading Light: Together these shadows and highlights give an object a three-dimensional look.
  1. Highlight
  2. Transitional Light
  3. Core of the Shadow
  4. Reflected Light
  5. Cast Shadow
- Linear Perspective: A concept relating to how an object seems smaller the farther away it gets.
- Atmospheric Perspective: Based on how air acts as a filter to change the appearance of distance objects.

FORM

Form is any three dimensional object. Form can be measured, from top to bottom (height), side to side (width), and from back to front (depth). Form is also defined by light and dark. There are two types of form, geometric (man-made) and natural (organic form). Form may be created by the combining of two or more shapes. It may be enhanced by tone, texture and color. It can be illustrated or constructed.

FORM FOLLOWS FUNCTION.

Originally a principle associated with modern architecture and industrial design in the 20th century, the concept is

now used more widely as an exhortation to base the form on the required functional use, and avoid ornamentation.

PRINCIPLES OF DESIGN

Principles applied to the elements of design that bring them together into one design. How one applies these principles determines how successful a design may be.

UNITY

According to Alex White, author of *The Element of Graphic Design*, to achieve visual unity is a main goal of graphic design. When all elements are in agreement, a design is considered unified. No individual part is viewed as more important than the whole design. A good balance between unity and variety must be established to avoid a chaotic or a lifeless design. Ways to achieve unity

- Proximity: Elements that are physically close, are considered related.
- Similarity: Elements that are related should share similar position, size, color, shape, or texture.
- Repetition and Rhythm: Recurring position, size, color, and use of a graphic element shows unity. When the repetition has a focal point interruption it is considered rhythm.
- Theme with variation: Altering the basic theme achieves unity and helps keep interest.

POINT, LINE, AND PLANE (PLP)

PLP are the three most basic shapes in visual design and a good design contains all three. The key to using PLP is making the shapes overlap and share elements.

- Point: In design, a point can be the smallest unit of marking not simply a dot. Additionally, a point can be a small plane or a short line.
- Line: The trace of a point in motion, a thin stroke, or even a narrow plane can be considered a line. Typed text automatically creates visual lines.
- Plane: A plane can be perceived as a trace of a line in motion like dragging a piece of chalk across a blackboard sideways (long side down). Wide lines and large points may also create a plane.

BALANCE

It is a state of equalized tension and equilibrium, which may not always be calm. A unified design is also visually balanced so that no space takes away from the whole.

Types of balance

- Symmetrical: A formal balance is a mirror image of one half of the picture. It is vertically centered, static, and evokes a feeling of class or formality. The objects in each half of the mirror image may not be identical, but may be mirror images in sense of color, number of objects or any other element of design.
- Asymmetrical: An informal balance that is attention attracting and dynamic. It balances a number of items of smaller size on one side with a larger one on the other. The modern feel an asymmetrical design is complex to create as it takes skills to distribute the blank space.
- Radial: Balance arranged around a central element. The elements placed in a radial balance seem to 'radiate' out from a central point in a circular fashion.

- Overall: This mosaic form of balance normally arises from too many elements being put on a page. Due to the lack of hierarchy and contrast, this form of balance can look noisy.

HIERARCHY

A good design contains elements that lead the reader through each element in order of its significance. The type and images should be expressed starting from most important to the least.

SCALE

Using the relative size of elements against each other can attract attention to a focal point. When elements are designed larger than life, scale is being used to show drama.

DOMINANCE

Dominance is created by contrasting size, positioning, color, style, or shape. The focal point should dominate the design with scale and contrast without sacrificing the unity of the whole.

SIMILARITY AND CONTRAST

Some key aspects of a well designed document include dramatic contrasts, scrupulous similarity, and active white space. Planning a consistent and similar design is an important aspect of a designers work to make their focal point visible. Too much similarity is boring but without similarity important elements will not exist. Also, without contrast an image is uneventful so the key is to find the balance between similarity and contrast.

Ways to Develop a Similar Environment

- Keep it simple and eliminate clutter. Do not fill white spaces with garbage.
- Build a unique internal organization structure.
- Manipulate shapes of images and text to correlate together.
- Express continuity from page to page (in publications). Items to watch include headers, themes, borders, and spaces.
- Develop a style manual and stick with the format.

Ways to Create Contrast

- Space
- Filled vs Empty
- Near vs Far
- 2-D vs 3-D
- Position
- Top vs Bottom
- Isolated vs Grouped
- Centered vs Off Centre
- Form
- Simple vs Complex
- Beauty vs Ugly
- Whole vs Broken
- Direction
- Vertical vs Horizontal
- Stability vs Movement
- Convex vs Concave
- Structure

- Organized vs Chaotic
- Serif vs Sans Serif
- Mechanical vs Hand Drawn
- Size
- Big vs Little
- Long vs Short
- Deep vs. Shallow
- Color
- Grayscale vs Color
- Light vs Dark
- Warm vs Cool
- Texture
- Fine vs Coarse
- Smooth vs Rough
- Sharp vs Dull
- Density
- Transparent vs Opaque
- Thick vs Thin
- Liquid vs Solid
- Gravity
- Light vs Heavy
- Stable vs Unstable

Movement is the path the viewer's eye takes through the artwork, often to focal areas. Such movement can be directed along lines edges, shape and color within the artwork.

WEBSITE ARCHITECTURE

A website, also written as Web site, web site, or simply site, is a set of related web pages containing content such

as text, images, video,audio, etc. A website is hosted on at least one web server, accessible via a network such as the Internet or a private local area network through an Internet address known as a Uniform Resource Locator. All publicly accessible websites collectively constitute the World Wide Web.

A webpage is a document, typically written in plain text interspersed with formatting instructions of Hypertext Markup Language (HTML, XHTML). A webpage may incorporate elements from other websites with suitable markup anchors.

Webpages are accessed and transported with the Hypertext Transfer Protocol (HTTP), which may optionally employ encryption (HTTP Secure, HTTPS) to provide security and privacy for the user of the webpage content. The user's application, often a web browser, renders the page content according to its HTML markup instructions onto a display terminal.

The pages of a website can usually be accessed from a simple Uniform Resource Locator (URL) called the web address. The URLs of the pages organize them into a hierarchy, although hyperlinking between them conveys the reader's perceived site structure and guides the reader's navigation of the site which generally includes a home page with most of the links to the site's web content, and a supplementary about, contact and link page.

Some websites require a subscription to access some or all of their content. Examples of subscription websites include many business sites, parts of news websites, academic

journal websites, gaming websites, file-sharing websites, message boards, web-based email, social networkingwebsites, websites providing real-time stock market data, and websites providing various other services (e.g., websites offering storing and/or sharing of images, files and so forth).

HISTORY

The World Wide Web (WWW) was created in 1990 by CERN physicist Tim Berners-Lee. On 30 April 1993, CERN announced that the World Wide Web would be free to use for anyone. Before the introduction of HTML and HTTP, other protocols such as File Transfer Protocol and the gopher protocol were used to retrieve individual files from a server. These protocols offer a simple directory structure which the user navigates and chooses files to download. Documents were most often presented as plain text files without formatting, or were encoded in word processorformats.

OVERVIEW

Organized by function, a website may be

- a personal website
- a commercial website
- a government website
- a nonprofit organization website.

It could be the work of an individual, a business or other organization, and is typically dedicated to some particular topic or purpose. Any website can contain a hyperlink to

any other website, so the distinction between individual sites, as perceived by the user, may sometimes be blurred.

Websites are written in, or dynamically converted to, HTML (Hyper Text Markup Language) and are accessed using a software interface classified as a user agent. Web pages can be viewed or otherwise accessed from a range of computer-based and Internet-enabled devices of various sizes, including desktop computers, laptops, PDAs and cell phones.

A website is hosted on a computer system known as a web server, also called an HTTP server, and these terms can also refer to the software that runs on these systems and that retrieves and delivers the web pages in response to requests from the website users. Apache is the most commonly used web server software (according to Netcraft statistics) and Microsoft's IIS is also commonly used. Some alternatives, such as Lighttpd, Hiawatha or Cherokee, are fully functional and lightweight.

STATIC WEBSITE

A static website is one that has web pages stored on the server in the format that is sent to a client web browser. It is primarily coded in Hypertext Markup Language (HTML).

Simple forms or marketing examples of websites, such as *classic website*, a *five-page website* or a *brochure website* are often static websites, because they present pre-defined, static information to the user. This may include information about a company and its products and services through text, photos, animations, audio/video and interactive menus and navigation.

This type of website usually displays the same information to all visitors. Similar to handing out a printed brochure to customers or clients, a static website will generally provide consistent, standard information for an extended period of time. Although the website owner may make updates periodically, it is a manual process to edit the text, photos and other content and may require basic website design skills and software.

In summary, visitors are not able to control what information they receive via a static website, and must instead settle for whatever content the website owner has decided to offer at that time.

They are edited using four broad categories of software:

- Text editors, such as Notepad or TextEdit, where content and HTML markup are manipulated directly within the editor program
- WYSIWYG offline editors, such as Microsoft FrontPage and Adobe Dreamweaver (previously Macromedia Dreamweaver), with which the site is edited using a GUI interface and the final HTML markup is generated automatically by the editor software
- WYSIWYG online editors which create media rich online presentation like web pages, widgets, intro, blogs, and other documents.
- Template-based editors, such as RapidWeaver and iWeb, which allow users to quickly create and upload web pages to a web server without detailed HTML knowledge, as they pick a suitable template from a palette and add pictures and text to it in a desktop

publishing fashion without direct manipulation of HTML code.

DYNAMIC WEBSITE

A dynamic website is one that changes or customizes itself frequently and automatically, based on certain criteria.

Dynamic websites can have two types of dynamic activity: Code and Content. Dynamic code is invisible or behind the scenes and dynamic content is visible or fully displayed.

DYNAMIC CODE

The first type is a web page with dynamic code. The code is constructed dynamically on the fly using active programming language instead of plain, static HTML.

A website with dynamic code refers to its construction or how it is built, and more specifically refers to the code used to create a single web page. A dynamic web page is generated on the fly by piecing together certain blocks of code, procedures or routines. A dynamically generated web page would recall various bits of information from a database and put them together in a pre-defined format to present the reader with a coherent page. It interacts with users in a variety of ways including by reading cookies recognizing users' previous history, session variables, server side variables etc., or by using direct interaction (form elements, mouse overs, etc.). A site can display the current state of a dialogue between users, monitor a changing situation, or provide information in some way personalized to the requirements of the individual user.

DYNAMIC CONTENT

The second type is a website with dynamic content displayed in plain view. Variable content is displayed dynamically on the fly based on certain criteria, usually by retrieving content stored in a database.

A website with dynamic content refers to how its messages, text, images and other information are displayed on the web page, and more specifically how its content changes at any given moment. The web page content varies based on certain criteria, either pre-defined rules or variable user input. For example, a website with a database of news articles can use a pre-defined rule which tells it to display all news articles for today's date. This type of dynamic website will automatically show the most current news articles on any given date. Another example of dynamic content is when a retail website with a database of media products allows a user to input a search request for the keyword Beatles. In response, the content of the web page will spontaneously change the way it looked before, and will then display a list of Beatles products like CDs, DVDs and books.

SOFTWARE SYSTEMS

There is a wide range of software systems, such as ANSI C servlets, JavaServer Pages (JSP), the PHP, Perl, Python, and Ruby programming languages, ASP.NET, Active Server Pages (ASP), YUMA and ColdFusion (CFML) that are available to generate dynamic web systems and dynamic sites. Sites may also include content that is retrieved from one or more databases or by usingXML-based technologies such as RSS.

Static content may also be dynamically generated either periodically, or if certain conditions for regeneration occur (cached) in order to avoid the performance loss of initiating the dynamic engine on a per-user or per-connection basis.

Plug ins are available to expand the features and abilities of web browsers to show *active content* or even create rich Internet applications. Examples of such plug-ins are Microsoft Silverlight,Adobe Flash, Adobe Shockwave or applets written in Java. Dynamic HTML also provides for user interactivity and realtime element updating within web pages (i.e., pages don't have to be loaded or reloaded to effect any changes), mainly using the Document Object Model (DOM) and JavaScript, support which is built-in to most modern web browsers.

Turning a website into an income source is a common practice for web developers and website owners. There are several methods for creating a website business which fall into two broad categories, as defined below.

CONTENT-BASED SITES

Some websites derive revenue by selling advertising space on their site either through direct sales or through an advertising network.

PRODUCT- OR SERVICE-BASED SITES

Some websites derive revenue by offering products or services for sale. In the case of e-commerce websites, the products or services may be purchased at the website itself, by entering credit card or other payment information into a payment form on the site. While most business websites

serve as a shop window for existing brick and mortar businesses, it is increasingly the case that some websites are businesses in their own right; that is, the products they offer are only available for purchase on the web.

Websites occasionally derive income from a combination of these two practices. For example, a website such as an online auctions website may charge the users of its auction service to list an auction, but also display third-party advertisements on the site, from which it derives further income.

SPELLING

The form "website" has become the most common spelling, but "Web site" (capitalised) and "web site" are also widely used, though declining. Some academia, some large book publishers, and some dictionaries still use "Web site", reflecting the origin of the term in the proper name World Wide Web. There has also been similar debate regarding related terms such as web page, web server, and webcam.

Among leading style guides, the Reuters style guide, *The Chicago Manual of Style*, and the *AP Stylebook* (since April 2010) all recommend "website".

Among leading dictionaries and encyclopedias, the *Canadian Oxford Dictionary* prefers "website", and the *Oxford English Dictionary* changed to "website" in 2004. Wikipedia also uses "website", but Encyclopædia Britannica (including its Merriam-Webster subsidiary) uses "Web site".

Among leading language-usage commentators, *Garner's Modern American Usage* acknowledges that "website" is the

standard form, but Bill Walsh, of *The Washington Post*, argues for using "Web site" in his books and on his website (however, *The Washington Post* itself uses "website").

Among major Internet technology companies and corporations, Google uses "website", as does Apple, though Microsoft uses both "website" and "web site".

TYPES OF WEBSITES

Websites can be divided into two broad categories - static and interactive. Interactive sites are part of the Web 2.0 community of sites, and allow for interactivity between the site owner and site visitors. Static sites serve or capture information but do not allow engagement with the audience directly. Some websites may be included in one or more of these categories. For example, a business website may promote the business's products, but may also host informative documents, such aswhite papers. There are also numerous sub-categories to the ones listed above. For example, a porn site is a specific type of e-commerce site or business site (that is, it is trying to sell memberships for access to its site) or have social networking capabilities. A fansite may be a dedication from the owner to a particular celebrity.

Websites are constrained by architectural limits (e.g., the computing power dedicated to the website). Very large websites, such as Facebook, Yahoo!, Microsoft, and Google employ many servers and load balancing equipment such as Cisco Content Services Switches to distribute visitor loads over multiple computers at multiple locations. As of early 2011, Facebook utilized 9 data centers with

approximately 63,000 servers. In February 2009, Netcraft, an Internet monitoring company that has tracked Web growth since 1995, reported that there were 215,675,903 websites with domain names and content on them in 2009, compared to just 18,000 websites in August 1995.

AWARDS

The Webby Awards, Favourite Website Awards, Interactive Media Awards and WebAwards are prominent award organizations recognizing the world's best websites.

WEBSITE MONETIZATION

Website monetization is the process of converting existing traffic being sent to a particular website into revenue. The most popular ways of monetizing a website are by implementing Pay per click (PPC) and Cost per impression (CPI/CPM) advertising. Various ad networks facilitate a webmaster in placing advertisements on pages of the website to benefit from the traffic the site is experiencing.

PAY PER CLICK ADVERTISING

Pay per click (also called Cost per click) is a marketing strategy put in place by search engines and various Advertising networks, where an advert, usually targeted by keywords or general topic, is placed on a relevant website. The advertiser then pays for every click that is made on the advert.

COST PER IMPRESSION ADVERTISING

Cost per impression (also called Cost per mille) is a marketing strategy put in place by various Advertising

networks, where an advert is placed on a relevant website, usually targeted to the content sector of that site. The advertiser then pays for every time the advert is displayed to a user.

BANNER ADVERTISING

Banner advertising consists of placing a graphical banner advertisement on a webpage. The role of this banner is to catch the eye of incoming traffic to the page, enticing readers to click the advertisement. This form of monetization is implemented by both affiliate programs and advertising networks. Banners originally just referred to advertisements of 468 x 60 pixels, but the term is now widely used to refer to all sizes of display advertising on the internet.

BANNER AD TYPES

Banner ads come in various shapes and sizes and are sized according to pixel dimensions. Typical banner sizes include:

- Leaderboard 728 x 90
- Banner 468 x 60
- Skyscraper 120 x 600
- Wide Skyscraper 160 x 600

AFFILIATE PROGRAMS

Affiliate programs are another popular way of monetizing existing website traffic. By joining a business' affiliate program, any searches for products within that business' catalogue may earn affiliates a commission on each sale that was originally referred through their website.

DATA MONETIZATION

Websites also generate valuable user data that can be monetized through various methods. Data generated by websites about their users can range from being demographics to in-market data (i.e. in-market for a car). This data can be sold through behavioral data exchanges and used by advertisers to target their online media campaigns. Websites can also generate revenue from their newsletter and on-site registrations programs by helping to bring in offline data associated with users during this process.

PAID MEMBERSHIP PROGRAMS

Membership or Continuity Programs Paid membership programs are another way to monetize existing traffic. Probably the most well known media membership sites are the Wall Street Journal and the New York Times. In the gaming world, Blizzard's World of Warcraft has millions of members. But there are many other kinds of member sites that cover niche markets. Often people join to get access to content and expertise, or for community, such as discussion or bulletin boards. The term "continuity" is used because the goal is to develop income continuity. Instead of making a one-time sale of a product or service, the membership site brings new, repeated income every month.

Besides news, other kinds of membership site include: health, fitness, marketing, copy writing, social media expertise, paper products, dating, paper crafting, scrap booking, coaching, writing and many other applications.

Experts in the membership site field say that "people come for content and stay for community." The challenge

of a member site is to retain paying members. Some sites, like the New York Times, offers some free content and then, charges a fee for more in depth access, or access to special kinds of content. Some sites offer downloads of audio or video content, free graphics, free software that is only available to members. Many sites also offer webinars to members. The webinars are often recorded as video, audio and also transcribed, creating more special content that's behind the pay wall.

Fees for membership vary widely. They can be billed monthly, annually, or even lifetime memberships. The digital access to the website is sometimes sold as part of a combination package that also includes physical product. For example, the Wall Street Journal offers a combination paper subscription, which is delivered to the subscriber's door, combined with access to the website and the smartphone app versions of the paper for about $140. Another site that sells membership to large corporations in the mobile phone industry, charges up to $12,000.00 a year for membership, which gives tech employees the right to pay to attend conferences on different aspects of the technology of cellular phones, and to access, on the website, recordings of past meetings.

Business sites may offer a special information package, perhaps CDs or DVDs shipped to the new member as part of a package that includes membership. Affiliate marketing is sometimes used to build membership in membership sites. Some sites continue to pay a percentage to the referring affiliate as long as the member continues paying monthly fees. Others pay a larger up front fee.

The page that marketers use a marketing or social media "funnel" to bring potential new paying members to is called a "squeeze" page. There is an annual Continuity Summit meeting organized by Ryan Lee that brings together experts in member sites.

WEBSITE WIREFRAME

A website wireframe, also known as a page schematic or screen blueprint, is a visual guide that represents the skeletal framework of a website. Wireframes are created by User Experience professionals called Interaction Designers. The interaction designers who have broad backgrounds in visual design, information architecture and user research, create wireframes for the purpose of arranging elements to best accomplish a particular purpose. The purpose is usually being informed by a business objective and a creative idea. The wireframe depicts the page layout or arrangement of the website's content, including interface elements and navigational systems, and how they work together. The wireframe usually lacks typographic style, color, or graphics, since the main focus lies in functionality, behaviour, and priority of content. In other words, it focuses on "what a screen does, not what it looks like." Wireframes can be pencil drawings or sketches on a whiteboard, or they can be produced by means of a broad array of free or commercial software applications.

Wireframes focus on

- The kinds of information displayed
- The range of functions available
- The relative priorities of the information and functions

- The rules for displaying certain kinds of information
- The effect of different scenarios on the display

The website wireframe connects the underlying conceptual structure, or information architecture, to the surface, or visual design of the website. Wireframes help establish functionality, and the relationships between different screen templates of a website. An iterative process, creating wireframes is an effective way to make rapid prototypes of pages, while measuring the practicality of a design concept. Wireframing typically begins between "high-level structural work—like flowcharts or site maps—and screen designs." Within the process of building a website, wireframing is where thinking becomes tangible.

Aside from websites, wireframes are utilized for the prototyping of mobile sites, computer applications, or other screen-based products that involve human-computer interaction. Future technologies and media will force wireframes to adapt and evolve.

USES OF WIREFRAMES

Wireframes may be utilized by different disciplines. Developers use wireframes to get a more tangible grasp of the site's functionality, while designers use them to push the user interface (UI) process. User experience designers and information architects use wireframes to show navigation paths between pages. Business stakeholders use wireframes to ensure that requirements and objectives are met through the design. Other professionals who create wireframes include information architects, interaction designers, user experience designers, graphic designers, programmers, and product

managers. Working with wireframes may be a collaborative effort since it bridges the information architecture to the visual design. Due to overlaps in these professional roles, conflicts may occur, making wireframing a controversial part of the design process. Since wireframes signify a "bare bones" aesthetic, it is difficult for designers to assess how closely the wireframe needs to depict actual screen layouts.

Another difficulty with wireframes is that they don't effectively display interactive details. Modern UI design incorporates various devices such as expanding panels, hover effects, and carousels that pose a challenge for 2-D diagrams. Wireframes may have multiple levels of detail and can be broken up into two categories in terms of fidelity, or how closely they resemble the end product. Low-fidelity Resembling a rough sketch or a quick mock-up, low-fidelity wireframes have less detail and are quick to produce. These wireframes help a project team collaborate more effectively since they are more abstract, using rectangles and labeling to represent content. Dummy content, Latin filler text (lorem ipsum), sample or symbolic content are used to represent data when real content is not available.

High-fidelity High-fidelity wireframes are often used for documenting because they incorporate a level of detail that more closely matches the design of the actual webpage, thus taking longer to create.

For simple or low-fidelity drawings, paper prototyping is a common technique. Since these sketches are just representations, annotations—adjacent notes to explain behaviour–are useful. For more complex projects, rendering wireframes using computer software is popular. Some tools

allow the incorporation of interactivity including Flash animation, and front-end web technologies such as, HTML, CSS, and JavaScript.

ELEMENTS OF WIREFRAMES

The skeleton plan of a website can be broken down into three components: information design, navigation design, and interface design. Page layout is where these components come together, while wireframing is what depicts the relationship between these components.

INFORMATION DESIGN

Information design is the presentation—placement and prioritization of information in a way that facilitates understanding. Information design is an area of graphic design, meant to display information effectively for clear communication. For websites, information elements should be arranged in a way that reflects the goals and tasks of the user.

NAVIGATION DESIGN

The navigation system provides a set of screen elements that allow the user to move page to page through a website. The navigation design should communicate the relationship between the links it contains so that users understand the options they have for navigating the site. Often, websites contain multiple navigation systems such as a global navigation, local navigation, supplementary navigation, contextual navigation, and courtesy navigation.

INTERFACE DESIGN

User interface design includes selecting and arranging interface elements to enable users to interact with the functionality of the system. The goal is to facilitate usability and efficiency as much as possible. Common elements found in interface design are action buttons, text fields, check boxes, radio buttons and drop-down menus.

WEB DESIGN DEFINITION

What is Web design? The definition of "Web Design" can vary, depending on who you ask. Web designers working for one company may perform different tasks than Web designers working for another company. The basic answer is that Web design is the design of a Web page or Website, including the information and user interface design, but not including programming. Programming falls under the definition of Web development, or Web application programming (to name two of many).

At a smaller company, with fewer people and more overlap of job descriptions, Web design can be defined as the whole production of the Website from start to finish. To clarify this a bit, let's outline the process of creating a Website from scratch.

Discovery: In this step, the Web designer finds out as much about the company and its clients as possible, paying special attention to the user audience of the Website.

Planning: Project definition documents are created as a guide to the creation of the Website. It is important that the scope, audience and goals of the Website are clearly

defined during this stage, so the resulting project definition can be used as a touchstone to keep everyone on track throughout the process.

Information Design: How will the information be broken down and presented to the user? If what the user will be looking for is well defined in the discovery and planning stages, this will be an easier job. The information design, or information architecture, step includes design of the navigation and is the most critical step in making the Website user-friendly.

Graphic Design: Graphic design may seem trivial to some, but it is also a very important factor in the usability of the Website. It isn't just about making the Website look pretty. It is also about visual balance and readable typography, both of which are critical in the creation of a user-friendly Web design.

When these steps have been completed, you have a finished Web design. Loosely speaking, putting it together is called Web production, and making it work is called Web development.

5

Multidisciplinary Design and Optimization

Multidisciplinary design and optimization is as the name implies, the process of combining a full set of computational design tools to create an optimum design. The process is necessarily iterative in nature and all of the disciplines normally utilized in an aircraft design are computationally intensive. An MDO approach for an aircraft could include aerodynamics, structures, and systems Computer Aided Engineering (CAE) tools.

Initial design assumptions would be input to each CAE toolset and the constraints and parameters to be optimized defined. Each CAE suite would then compute design parameters that would be utilized by the other CAE tools as a subset of their required inputs. The ultimate design would theoretically be structurally sounder, lighter and more cost effective to fabricate.

The design timeframe would be also very much shortened. The challenges to this process are in the exchange of data between the CAE applications and the tuning of the entire process to achieve convergence on the final solution set in an efficient manner.

STRUCTURAL ANALYSIS

The optimization of analytical design tools is a process that will lead to shortened design time frames, lighter and more efficient designs, with reduced production and life cycle costs of the final design. The many analytical tools now available have been typically developed for specific applications and are often not readily applicable outside of their original design target arena.

An example lies in the structural analysis field where tools developed for metallics will be much different from those developed for composite materials where material properties may vary according to axis.

The ability to rapidly define an optimized aircraft structure having light weight, and improved fatigue and damage tolerance capabilities, is a critical technology to maintain competitive leadership in the development and supply of future new aircraft.

This will be achieved by the extensive use of computerized methods for structural analysis and design optimization, and the analysis of failure and fracture mechanics.

The methods must be integrated with the in-house design and manufacturing data bases, the 3-D CAD/CAMsystems, and also be easy to use. Suppliers and partners will have access to the resulting design information via Technical Data Interchange (TDI). This will ensure consistency with an up-

to-date knowledge of the requirements for loads, interfaces and the space envelopes available for their products. The immediate dissemination to suppliers of information on design changes will help diminish subsequent redesign activity and the time and cost penalties incurred for rework.

The preliminary structural design will often use detailed Finite Element Methods (FEM) for analysis, coupled with constrained optimization, and the process must be highly automated for rapid creation of FEM meshing for models. In order to achieve shortened design cycle time, the loads and dynamics stiffness requirements must become available much sooner than at present. This will require early development of MDO models for overall aerodynamic and structural optimization that will define the static and dynamic loads for flight and ground operations. Trade-off studies must rapidly search for the best designs and arrive at realistic structural sizes, providing space envelopes and accurate weights to minimize subsequent redesign.

STRUCTURAL DESIGN, ANALYSIS AND OPTIMIZATION

Shortened design cycle times are necessary for achieving market advantage in the aerospace and defence sector. Improvements in the structural analysis, design and optimization of gas turbine engines is necessary to achieve these goals while also meeting the overall objectives of increased durability and efficiency at lower costs. A Multi-disciplinary Design Optimization (MDO) approach that combines finite element analysis and aerodynamic design techniques is employed. MDO is necessary to rapidly

determine the structure of the engine and identify critical areas requiring further or more detailed analysis.

Many of the structural and aerodynamic codes developed by companies are proprietary in nature and the integration and refinement of these codes is an on-going challenge.

COMPUTATIONAL FLUID DYNAMICS

COMPUTATIONAL DEVELOPMENT AND VALIDATION

Computational Fluid Dynamics (CFD) has had the greatest effect on both aircraft and engine design of any single design tool over the past twenty-five years. Computational power and cost have enabled widespread application and development of CFD techniques. Computational fluid dynamics is basically the use of computers to numerically model flows of interest. Nodes in the flowpath are identified and equations of motion solved at these locations to identify flow parameters.

In essence a grid or mesh is defined over the surface of the object that extends outwards into the flowfield containing the object. Flow equations are then calculated at each node in the grid, and iteratively re-calculated until all results for each node are within an acceptable variance.

The equations used are either Euler based which do not include viscous effects (boundary layers) directly, or Navier-Stokes equations which include viscous effects and which produce more accurate but computationally more demanding solutions. Such methods can be used for external flows about an aircraft or for internal flows in a gas turbine including

combustion. The Euler based analyses are typically less computationally demanding but are less precise for modeling separated flows on wings and bodies, or for internal reversed flows. It should be noted that Navier first developed his equations in 1823 and that Stokes refined them in 1845. The development of solutions to these equations was not feasible until the latter part of this century. Today much R&D effort on NS methods is expended on improving modeling of the turbulent flow terms for specific problems.

Numerous forms of Euler and Navier-Stokes solutions have been developed to address particular design problems. Solutions to these equations are dependent on experimentation for both coefficients and for validation.

Mesh selection and node placement is critical to the solution of the flowfield. The automated generation of meshes is now in wide spread use and can often be linked to Computer Aided Engineering and Design tools. The form of the equation used, the density of the mesh or grid and convergence requirements determine computational demands. Complete aircraft solutions require huge computer resources and much R&D is aimed at improving the speed of the solution.

COMPUTATIONAL FLUID DYNAMICS - GAS TURBINES

CFD is perhaps the single most critical technology for gas turbine engines. Gas turbineCFD needs have typically posed the greatest challenges to engine designers, and computational power and code developers. While CFD is of utmost importance to the engine designer it is a very specific disciplinary design requirement and competence is held by

a very small number of engine design firms worldwide. Computation techniques for gas turbine engines also tend to be very module specific — compressor, transition duct, combustor, turbine and exhaust duct/military afterburner are examples. Computational techniques are often also specific to engine size class and thus Canada, focusing on small gas turbines, has a specific set of technology requirements.

Advanced 3D CFD codes have been used to generate the following design improvements:

- In the compressor to develop advanced swept airfoils capable of high compression ratios that in turn yield higher efficiency at less weight and with a smaller parts count (significant life cycle cost factor);
- In the combustor for higher intensity (smaller volumes with much higher energy density) combustors that approach stoichiometric conditions to yield higher efficiency with lower weight; and
- In the turbine to produce higher stage loading with reduced turbine cooling air requirements that again reduces weight and cost while reducing fuel burn.

COMBUSTION SYSTEMS COMPUTATION

The combustor of a gas turbine engine is that part of the engine that receives the compressed air from the compressor. Energy is added to the airflow in the combustor in the form of chemical energy derived from fuel. The combustor discharge air is expanded across a turbine or turbines where energy is extracted to drive the compressor and gearbox of a turboshaft/turboprop engine, or to provide jet thrust via a

turbofan and core nozzle in a thrust engine.

Small gas turbines, of the size that have typically been designed and built in Canada pose significant design challenges because of their size. Pratt and Whitney Canada combustors are the highest intensity combustors in the world, where intensity can be thought of as the amount of energy converted per unit volume within the combustor.

The design objectives for gas turbine engines, including small ones, are to increase both overall pressure ratios and cycle temperatures, which lead to increased efficiency and smaller size and weight, while simultaneously producing reduced noise and noxious emissions levels. Combustor technology development challenges for Canadian engine manufacturers include.

- *Computational fluid dynamics*: CFD analyses are complicated by the reverse flow designs typically selected to maintain short combustors within small volumes. Cooling flow and chemical additions to the CFD design further complicate the process as the temperatures of gases at the core of the flows are well above the melting temperatures of the combustor materials. Pressure losses and cooling flow requirements must be minimized to improve performance.
- *Materials*: Increasing compressor ratios result in increased compressor discharge temperatures and decreased cooling capability. These increased temperatures also push for higher fuel to air ratios and higher temperatures within the combustor. Stoichiometric ratio is that ratio when all oxygen is consumed in the combustion process leaving less air

for cooling. Materials challenges in this environment are the most demanding. Fuel injection and mixing: CFD and injector specific techniques are required.

- *Emissions*: While not legislated and not contributing significantly in absolute terms, there is a drive for lower emissions that drives designs often in the opposite direction to those factors identified above.

AERODYNAMICS AND FLIGHT MECHANICS

Aerodynamics is the study of forces on wing bodies and controls due to air pressure and viscous (drag) effects. Flight mechanics is the study of the resulting motion of objects through the air and includes the stability and control Behaviour. The laws of motion and aerodynamics are combined to ensure that an aircraft flies in the intended manner. Much of the aerodynamics and flight mechanics work that is pursued for the purposes of aircraft designed and built in Canada will pertain to such issues as the design of improved wings, the integration of various components onto an aircraft or issues such as flight in adverse conditions where the handling qualities of an aircraft will be adversely influenced by the build up of ice on the surface of the wing.

Advanced technology development in this field will be directed towards supersonic transports and eventually hypersonic flight. There are considerable differences between fixed wing and rotary wing aircraft aerodynamics and flight mechanics and both areas are of considerable interest to the Canadian aerospace and defence industry. Technologies relevant to Aerodynamics and Flight Mechanics are described below,

ADVANCED AERODYNAMICS AND HANDLING

Included here are technologies that will enable the Canadian Aerospace industry to contribute to the design of advanced concept aircraft technologies or components or be the lead design integrator.

These enabling technologies should be pursued dependent on their links to, and pre-positioning for potential application to specific aircraft platforms or types as follows:

- *Future Transport Aircraft*: Future transport aircraft will have to demonstrate increased speed and load carrying capabilities over greatly extended ranges. Specific targets have been set by the U.S. for next generation transport aircraft although no new advanced concept transport aircraft are currently well advanced. Wing loading factors will double over that of existing aircraft with the development of materials new to the transport aircraft envelope. For shorter-range aircraft, a key enabling technology will be that of high efficiency turboprop engines with cruise speeds above the M.72 range. Propulsion technology and propulsion integration issues, aircraft design optimization, CFD, and materials technology development and insertion will be key to the success of the future transport aircraft.
- *Hypersonic Aircraft*: Hypersonic aircraft are in exploratory or advanced development model stage at this time and will be used initially for low cost space launch and delivery platforms and subsequently for commercial transport. Propulsion technologies are significant to hypersonic vehicle feasibility and are now the limiting

factor. Variable cycle engines, advanced materials, endothermic fuels and fuel control technologies are key aeropropulsion technology elements where significantR&D remains unsatisfied. Numerous controls and materials research topics require further investment as well, although less uncertainty remains in these areas due to advances made through the shuttle Programmes.

- *Advanced Rotorcraft*: Future rotorcraft will demonstrate increased cruise speeds of 200 kts or greater with tiltrotor speeds approaching 450 kts. These cruise speeds will be possible at significantly reduced vibration levels and with greatly increased range/fuel economy. Many of the design concepts for attaining these performance improvements are already in development, however much work remains undone.
- *Advanced Rotorcraft Flight Mechanics*: For both conventional helicopter and tiltrotor blades, the wings and propulsion system operate in a very complex aeromechanical environment. Aerodyn-amics, structures, vibration and acoustics parameters are inseparable and typically drive the design of the entire air vehicle. In trimmed forward flight the advancing blade tip will be moving at near sonic velocities whilst the retreating blade is often in near stall conditions.

ADVANCED DESIGN AND DEVELOPMENT

General aviation aircraft pose specific design challenges in all aspects of their design and fabrication. Increasing availability of low cost and high performance avionics, advanced composite designs and powerplant integration all

offer opportunities for general aviation aircraft designers and builders. Many of the technologies being furthered for use in military unmanned aerial vehicles will be of pertinence to general aviation aircraft. Low cost gas turbine technologies and composite structures development and certification issues will likely be the technologies of greatest interest.

The development of technologies for military purposes will underwrite some of the costs of introduction of those design concepts into general aviation use.

EXPERIMENTAL ASSESSMENT AND PERFORMANCE

Analytical design and analysis techniques are a prerequisite to reductions in design cycle time, design and production costs, and improved safety and environmental impact. The development of these analytical or numerical design techniques will remain heavily dependent on experimental validation of design codes and performance targets for another 10-15 years. Whereas in the past, experimental resources such as wind tunnels were used primarily for design development and refinement, in the future they may increasingly be used for the validation of computational design tools.

Notwithstanding the foregoing, there will continue to be a requirement for national facilities including wind tunnels, engine test facilities, flight test resources, and specialized resources including icing tunnels and rig test facilities for some time to come.

Experimental design and performance validation technology investment will be required in the following areas to support the aerospace industry in Canada:

- *Data Capture and Analysis Automation*: Automated methods for intelligent data capture and analysis will be required to reduce large facility run times and meet the challenges of design tool validation. This will require investment both in sensors and in computational tools;
- *Experimental Code Development*: Increased data capture rates and fidelity will be required and will necessitate the development of specific codes for experimental design and performance validation. Facilities and infrastructure will have to be maintained or enhanced to achieve these goals; and
- *Infrastructure Support*: The maintenance of critical national facilities will have to be supported in concert with other government departments and industry. The objective will not necessarily be to create new facilities but rather to improve the functionality of existing resources to meet the needs of new technology developments.

AEROPROPULSION PERFORMANCE ASSESSMENT

Test cells utilized for Canadian aero-engine Programmes, and also those developed for sale, have typically been sea-level static facilities offering little or no altitude, forward flight velocity or temperature pressure simulation. Some limited flying test bed capability exists in Canada for the testing of engines.

That being said, the National Research Council has participated in numerous international projects in the process ensuring that a world leading test cell capability exists

both for engine qualification testing, performance testing and for the development of performance assessment techniques.

Engine test cells take a number of forms. Sea level test facilities are used for Engine Qualification Testing that involves the monitoring of a relatively small number of parameters over long periods where in-service usage is evaluated in a time compressed manner.

Qualification testing also involves the ingestion of ice or water to ensure that unacceptable engine degradation does not occur in those instances. The NRC Institute for Aerospace Research has developed world recognized icing testing competencies and icing test facilities that are used by Canadian and off-shore engine manufacturers for qualification testing.

Altitude test cells are used to qualify engines over a full flight envelope as opposed to the endurance type testing previously described. The National Research Council in colLabouration with Pratt and Whitney Canada have developed and operated one small altitude test cell at NRC for some time.

An initiative that began in 2000 will see the development and commissioning of a somewhat larger and more capable altitude facility, again as a colLabourative effort between NRC and P&WC.

Test cells can also be used for the analysis of problems or validation of problem resolution. In these cases the test cells often require enhanced instrumentation suites and a much more careful design to ensure that performance parameters are correctly measured. World interest in advanced test cell technologies has been directed at those required to support

hypersonic vehicles for military uses or for space launch vehicles. This type of test cell is very resource intensive and highly specialized and will likely be of little interest or utility to any but a limited number of Canadian firms.

The Short Take Off and Vertical Landing (STOVL) version of the F35 Joint Strike Fighter has recently posed new challenges in the world of aeropropulsion testing. For this testing, in-flow preparation, exhaust treatment, fan drive systems, and 6 axis thrust measurement in the vertical axis will all pose significant new challenges to the performance assessment community.

ADVANCED D®ESIGN CONCEPTS

ANALYSIS AND DESIGN INTEGRATION

Advanced aerodynamics profile development in Canada will be primarily directed at wing design for subsonic aircraft carrying less than 120 passengers. The objective of work done on advanced aerodynamic profiles will be to increase efficiency and cruise speeds through reduced drag while improving structural and control characteristics. Wing profile, control surface effectiveness, airframe and engine interface effects with the wing and wing tip designs are areas of research and development interest. Also, developments improving wing-flap high lift performance are important areas for minimizing wing size required and hence costs.

Laminar flow control is a term that deserves discussion. Airflow over wings begins as a laminar or ordered flowfield and will transition to a higher drag producing turbulent flow based on flow characteristics such as speed and wing

influences including wing shape, surface roughness.

It has been estimated that if laminar flow could be maintained on the wings of a large aircraft, fuel savings of up to 25% could be achieved. Wing and flight characteristics of small aircraft are such that laminar flow can be relatively easily maintained over much of the flight envelope. A variety of methods can be used to increase laminar flow regions on aircraft of larger size and having higher Reynolds numbers and sweep angles. Computational fluid dynamics will be the most important technology relevant to the development of advanced aerodynamic profiles. A number of areas require R&D activity and support for aircraft design particular to Canadian aerospace interests. Large-scaleCFD code refinement and validation is one area requiring work to improve accuracy and reduce computational times for MDO by more rapid design convergence. These CFD codes will also require validation in Labouratories and in wind tunnels.

ALL-ELECTRIC AIRCRAFT CONCEPT DEVELOPMENT

The all-electric aircraft will utilize electronic actuators to replace equivalent hydraulic system components. The intent is to save weight and increase reliability. For example, electrical generators would provide power to electric actuators for flight control surface movement rather than equivalent hydraulic powered components. Electric power cables are lighter and less prone to damage or service induced degradation such as fitting vibration that results in leakage in hydraulic systems. Alternate power supply redundancy is an additional advantage of this concept. Challenges

associated with this type of technology insertion would be related to electromagnetic interference (EMI), and rapid load fluctuations imposed on the power generation engines.

FLY-BY-LIGHT CONCEPT DEVELOPMENT

Fly-by-Light (FBL) technology involves the replacement of electronic data transmission, mechanical control linkages, and electronic sensors with optical components and subsystems. Benefits include lower initial acquisition and life cycle costs, reduced weight, and increased aircraft performance and reliability. Fibre-optic cables are essentially immune to electroma-gnetic interference and therefore not affected by fields generated by other lines or electrical devices in close proximity, nor are they affected by lightning strikes. For flight controls, hydraulic or electric actuators are still employed but receive their command inputs via fibre-optic cables. Weight reductions are significant as the fibre-optic cables need only be protected from physical damage, whereas electric cables must be insulated and shielded increasing weight significantly. Also with a FBL connection multiple routes can be readily provided that are well separated to provide control redundancy.

There are a number of enabling technologies that must be developed in order to enable photonics technology insertion. Fibre-optic connectors for in-line and end connections must be developed that are durable and insensitive to in-service maintenance activities. Fibre-optic sensors development will also be necessary to allow the achievement of the full range of benefits that can be obtained in fly-by-light aircraft.

This technology is usually associated with smart structures

concepts such as smart skins where fibre-optic cabling can be readily embedded in a composite lay-up to achieve dispersed damage, stress, temperature or vibration sensing capability.

DETECTION MANAGEMENT AND CONTROL SYSTEMS

Regional airliners and helicopters operating in lower level airspace are increasingly exposed to hazardous icing conditions. This has increased the need for technologies for proactive and reactive ice detection and protection. Reactive technologies are those related to the detection of runback icing and attempt to monitor real-time or infer likely aerodynamic performance degradation.

Proactive systems forecast the potential for icing conditions and provide on-board avoidance advisory information. Reactive systems provide reasonable protection of the aircraft within the regulated flight envelope but are essentially go/no-go decision aids.

Aircraft on Search and Rescue Missions and most civil transport aircraft often do not have the option of avoiding hazardous icing conditions and should have pro-active pilot advisors and ice removal systems.

Reactive ice detection devices include: Embedded sensors that are mounted on the wing surface in a critical location and monitor ice build-up; and aerodynamic performance sensors that typically monitor pressure within the boundary layer of the wing to determine lift performance degradation.

Proactive systems require the remote measurement of Liquid Water Content (LWC), Outside Air Temperature (OAT)

and Mean Volume Diameter (MVD) of the liquid water. Knowledge of these three parameters is required to predict hazardous icing conditions. Additional R&D work on MVD measurement is required.

Ice control and removal systems may use heated air from the engines or electrical heat elements to remove ice from airfoil surfaces. Coatings that are termed"iceophobic" may also be applied to minimize ice build-up. CFD tools are needed to Analyse ice-buildup characteristics, assess aerodynamic degradation, and improve ice removal air supply performance.

This technology area is of particular interest because of the types of aircraft produced in Canada and because of climatic conditions.

DESIGN TECHNIQUES

Two issues predominate in the environmental design factors technology category for aeropropulsion systems — external noise and exhaust emissions.

A previously stated objective for noise reduction is in the order of 6 EPNdB (Effective Perceived Noise in dB). This objective can be achieved through the utilization of larger by-pass ratio fans, innovative design concepts for turbo fans and sound conscious designs in the combustor and exhaust nozzles/liners.

Generally speaking, noise improvements and fuel efficiency must be improved to meet future regulatory requirements without sacrifice of overall engine efficiency. Of special interest will be advanced ducted propulsors (ADF) that offer both noise attenuation and increased efficiency potential.

This technology area will be heavily dependent on

computational design techniques and multidisciplinary design optimization. The reduction in aircraft emissions is also a regulated requirement. While small aircraft engines contribute an insignificant amount of pollution they are still the targets of increased environmental scrutiny. Regulatory requirements are targeted at Nitrous Oxides (NOx), Carbon Monoxide (CO) and visible particulate emissions. CFD analysis techniques specific to combustion processes will be the major tool used to lower aeropropulsion emissions.

6

Analysis of Algorithms

PROGRAMMES

When analyzing a Programme in terms of efficiency, we want to look at questions such as, "How long does it take for the Programme to run?" and "Is there another approach that will get the answer more quickly?" Efficiency will always be less important than correctness; if you don't care whether a Programme works correctly, you can make it run very quickly indeed, but no one will think it's much of an achievement! On the other hand, a Programme that gives a correct answer after ten thousand years isn't very useful either, so efficiency is often an important issue.

The term "efficiency" can refer to efficient use of almost any resource, including time, computer memory, disk space, or network bandwidth. However, we will deal exclusively with time efficiency, and the major question that we want to ask about a Programme is, how long does it take to perform its

task? It really makes little sense to classify an individual Programme as being "efficient" or "inefficient." It makes more sense to compare two (correct) Programmes that perform the same task and ask which one of the two is "more efficient," that is, which one performs the task more quickly. However, even here there are difficulties. The running time of a Programme is not well-defined. The run time can be different depending on the number and speed of the processors in the computer on which it is run and, in the case of Java, on the design of the Java Virtual Machine which is used to interpret the Programme.

It can depend on details of the compiler which is used to translate the Programme from high-level language to machine language. Furthermore, the run time of a Programme depends on the size of the problem which the Programme has to solve. It takes a sorting Programme longer to sort 10000 items than it takes it to sort 100 items. When the run times of two Programmes are compared, it often happens that Programme A solves small problems faster than Programme B, while Programme B solves large problems faster than Programme A, so that it is simply not the case that one Programme is faster than the other in all cases.

In spite of these difficulties, there is a field of computer science dedicated to analyzing the efficiency of Programmes. The field is known as Analysis of Algorithms. The focus is on algorithms, rather than on Programmes as such, to avoid having to deal with multiple implementations of the same algorithm written in different languages, compiled with different compilers, and running on different computers. Analysis of Algorithms is a mathematical field that abstracts

away from these down-and-dirty details. Still, even though it is a theoretical field, every working Programmemer should be aware of some of its techniques and results. This is a very brief introduction to some of those techniques and results. Because this is not a mathematics book, the treatment will be rather informal.

One of the main techniques of analysis of algorithms is asymptotic analysis. The term "asymptotic" here means basically "the tendency in the long run." An asymptotic analysis of an algorithm's run time looks at the question of how the run time depends on the size of the problem. The analysis is asymptotic because it only considers what happens to the run time as the size of the problem increases without limit; it is not concerned with what happens for problems of small size or, in fact, for problems of any fixed finite size. Only what happens in the long run, as the problem size increases without limit, is important.

Showing that Algorithm A is asymptotically faster than Algorithm B doesn't necessarily mean that Algorithm A will run faster than Algorithm B for problems of size 10 or size 1000 or even size 1000000 — it only means that if you keep increasing the problem size, you will eventually come to a point where Algorithm A is faster than Algorithm B. An asymptotic analysis is only a first approximation, but in practice it often gives important and useful information.

Using this notation, we might say, for example, that an algorithm has a running time that is $O(n^2)$ or $O(n)$ or $O(\log(n))$. These notations are read "Big-Oh of n squared," "Big-Oh of n," and "Big-Oh of log n" (where log is a logarithm function). More generally, we can refer to $O(f(n))$ ("Big-Oh of f of n"),

where f(n) is some function that assigns a positive real number to every positive integer n. The "n" in this notation refers to the size of the problem.

Before you can even begin an asymptotic analysis, you need some way to measure problem size. Usually, this is not a big issue. For example, if the problem is to sort a list of items, then the problem size can be taken to be the number of items in the list. When the input to an algorithm is an integer, as in the case of an algorithm that checks whether a given positive integer is prime, the usual measure of the size of a problem is the number of bits in the input integer rather than the integer itself. More generally, the number of bits in the input to a problem is often a good measure of the size of the problem.

To say that the running time of an algorithm is O(f(n)) means that for large values of the problem size, n, the running time of the algorithm is no bigger than some constant times f(n). (More rigorously, there is a number C and a positive integer M such that whenever n is greater than M, the run time is less than or equal to C*f(n).) The constant takes into account details such as the speed of the computer on which the algorithm is run; if you use a slower computer, you might have to use a bigger constant in the formula, but changing the constant won't change the basic fact that the run time is O(f(n)).

The constant also makes it unnecessary to say whether we are measuring time in seconds, years, CPU cycles, or any other unit of measure; a change from one unit of measure to another is just multiplication by a constant. Note also that O(f(n)) doesn't depend at all on what happens for small

problem sizes, only on what happens in the long run as the problem size increases without limit.

To look at a simple example, consider the problem of adding up all the numbers in an array. The problem size, n, is the length of the array. Using A as the name of the array, the algorithm can be expressed in Java as:

```
total = 0;
for (int i = 0; i < n; i++)
total = total + A[i];
```

This algorithm performs the same operation, total = total + A[i], n times. The total time spent on this operation is a*n, where a is the time it takes to perform the operation once. Now, this is not the only thing that is done in the algorithm. The value of i is incremented and is compared to n each time through the loop. 4This adds an additional time of b*n to the run time, for some constant b. Furthermore, i and total both have to be initialized to zero; this adds some constant amount c to the running time. The exact running time would then be (a+b)*n+c, where the constants a, b, and c depend on factors such as how the code is compiled and what computer it is run on. Using the fact that c is less than or equal to c*n for any positive integer n, we can say that the run time is less than or equal to (a+b+c)*n.

That is, the run time is less than or equal to a constant times n. By definition, this means that the run time for this algorithm is O(n). If this explanation is too mathematical for you, we can just note that for large values of n, the c in the formula (a+b)*n+c is insignificant compared to the other term, (a+b)*n. We say that c is a "lower order term." When doing asymptotic analysis, lower order terms can be discarded. A rough, but correct, asymptotic analysis of the algorithm would

go something like this: Each iteration of the for loop takes a certain constant amount of time. There are n iterations of the loop, so the total run time is a constant times n, plus lower order terms (to account for the initialization). Disregarding lower order terms, we see that the run time is O(n).

Note that to say that an algorithm has run time O(f(n)) is to say that its run time is no bigger than some constant times f(n) (for large values of n). O(f(n)) puts an upper limit on the run time. However, the run time could be smaller, even much smaller. For example, if the run time is O(n), it would also be correct to say that the run time is $O(n^2)$ or even $O(n^{10})$. If the run time is less than a constant times n, then it is certainly less than the same constant times n^2 or n^{10}.

Of course, sometimes it's useful to have a lower limit on the run time. That is, we want to be able to say that the run time is greater than or equal to some constant times f(n) (for large values of n). The notation for this is Ω(f(n)), read "Omega of f of n." "Omega" is the name of a letter in the Greek alphabet, and Ω is the upper case version of that letter.

(To be technical, saying that the run time of an algorithm is Ω(f(n)) means that there is a positive number C and a positive integer M such that whenever n is greater than M, the run time is greater than or equal to C*f(n).) O(f(n)) tells you something about the maximum amount of time that you might have to wait for an algorithm to finish; Ω(f(n)) tells you something about the minimum time.

The algorithm for adding up the numbers in an array has a run time that is Ω(n) as well as O(n). When an algorithm

has a run time that is both Ω(f(n)) and O(f(n)), its run time is said to be Θ(f(n)), read "Theta of f of n." (Theta is another letter from the Greek alphabet.) To say that the run time of an algorithm is Θ(f(n)) means that for large values of n, the run time is between a*f(n) and b*f(n), where a and b are constants (with b greater than a, and both greater than 0). Let's look at another example. Consider the algorithm that can be expressed in Java in the following method:

```
/**
* Sorts the n array elements A[0], A[1], ..., A[n-1] into increasing order.
*/
    public static simpleBubbleSort(int[] A, int n) {
    for (int i = 0; i < n; i++) {
// Do n passes through the array...
    for (int j = 0; j < n-1; j++) {
        if (A[j] > A[j+1]) {
// A[j] and A[j+1] are out of order, so swap them
                int temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
    }
}
```

Here, the parameter n represents the problem size. The outer for loop in the method is executed n times. Each time the outer for loop is executed, the inner for loop is exectued n-1 times, so the if statement is executed n*(n-1) times. This is n^2-n, but since lower order terms are not significant in an asymptotic analysis, it's good enough to say that the if statement is executed about n^2 times.

In particular, the test A[j] > A[j+1] is executed about n^2times, and this fact by itself is enough to say that the run time of the algorithm is W(n^2), that is, the run time is at least

some constant times n^2. Furthermore, if we look at other operations — the assignment statements, incrementing i and j, etc. — none of them are executed more than n^2 times, so the run time is also $O(n^2)$, that is, the run time is no more than some constant times n^2. Since it is both $\Omega(n^2)$ and $O(n^2)$, the run time of the simpleBubbleSort algorithm is $\Theta(n^2)$.

You should be aware that some people use the notation O(f(n)) as if it meant $\Theta(f(n))$. That is, when they say that the run time of an algorithm is O(f(n)), they mean to say that the run time is about equal to a constant times f(n). For that, they should use $\Theta(f(n))$. Properly speaking, O(f(n)) means that the run time is less than a constant times f(n), possibly much less.

So far, the analysis has ignored an important detail. We have looked at how run time depends on the problem size, but in fact the run time usually depends not just on the size of the problem but on the specific data that has to be processed. For example, the run time of a sorting algorithm can depend on the initial order of the items that are to be sorted, and not just on the number of items.

To account for this dependency, we can consider either the worst case run time analysis or the average case run time analysis of an algorithm. For a worst case run time analysis, we consider all possible problems of size n and look at the longest possible run time for all such problems. For an average case analysis, we consider all possible problems of size n and look at the average of the run times for all such problems. Usually, the average case analysis assumes that all problems of size n are equally likely to be encountered,

although this is not always realistic — or even possible in the case where there is an infinite number of different problems of a given size.

In many cases, the average and the worst case run times are the same to within a constant multiple. This means that as far as asymptotic analysis is concerned, they are the same. That is, if the average case run time is O(f(n)) or Θ(f(n)), then so is the worst case. However, later in the book, we will encounter a few cases where the average and worst case asymptotic analyses differ.

So, what do you really have to know about analysis of algorithms to read the rest of this book? We will not do any rigorous mathematical analysis, but you should be able to follow informal discussion of simple cases such as the examples that we have looked. Most important, though, you should have a feeling for exactly what it means to say that the running time of an algorithm is O(f(n)) or Θ(f(n)) for some common functions f(n).

The main point is that these notations do not tell you anything about the actual numerical value of the running time of the algorithm for any particular case. They do not tell you anything at all about the running time for small values of n. What they do tell you is something about the rate of growth of the running time as the size of the problem increases. Suppose you compare two algorithms that solve the same problem. The run time of one algorithm is $\Theta(n^2)$, while the run time of the second algorithm is $\Theta(n^3)$. What does this tell you? If you want to know which algorithm will be faster for some particular problem of size, say, 100, nothing is certain. As far as you can tell just from the asymptotic

analysis, either algorithm could be faster for that particular case — or in any particular case.

But what you can say for sure is that if you look at larger and larger problems, you will come to a point where the $\Theta(n^2)$ algorithm is faster than the $\Theta(n^3)$ algorithm.

Furthermore, as you continue to increase the problem size, the relative advantage of the $\Theta(n^2)$ algorithm will continue to grow. There will be values of n for which the $\Theta(n^2)$ algorithm is a thousand times faster, a million times faster, a billion times faster, and so on. This is because for any positive constants a and b, the function $a*n^3$ grows faster than the function $b*n^2$ as n gets larger. (Mathematically, the limit of the ratio of $a*n^3$ to $b*n^2$ is infinite as n approaches infinity.)

This means that for "large" problems, a $\Theta(n^2)$ algorithm will definitely be faster than a $\Theta(n^3)$ algorithm. You just don't know — based on the asymptotic analysis alone — exactly how large "large" has to be. In practice, in fact, it is likely that the $\Theta(n^2)$ algorithm will be faster even for fairly small values of n, and absent other information you would generally prefer a $\Theta(n^2)$ algorithm to a $\Theta(n^3)$ algorithm.

So, to understand and apply asymptotic analysis, it is essential to have some idea of the rates of growth of some common functions. For the power functions n, n^2, n^3, n^4, ..., the larger the exponent, the greater the rate of growth of the function. Exponential functions such as 2^n and 10^n, where the n is in the exponent, have a growth rate that is faster than that of any power function.

In fact, exponential functions grow so quickly that an algorithm whose run time grows exponentially is almost certainly impractical even for relatively modest values of n,

because the running time is just too long. Another function that often turns up in asymptotic analysis is the logarithm function, log(n). There are actually many different logarithm functions, but the one that is usually used in computer science is the so-called logarithm to the base two, which is defined by the fact that $\log(2^x) = x$ for any number x. (Usually, this function is written $\log_2(n)$, but I will leave out the subscript 2, since I will only use the base-two logarithm in this book.) The logarithm function grows very slowly. The growth rate of log(n) is much smaller than the growth rate of n. The growth rate of n*log(n) is a little larger than the growth rate of n, but much smaller than the growth rate of n^2.

POINTS

- *Introduction*: Analysis of Selection Sort
- *Introduction*: Analysis of Merge Sort
- Asymptotic Notation
- Asymptotic Notation Continued
- Heapsort
- Heapsort Continued
- Priority Queues (more heaps)
- Quicksort
- Bounds on Sorting and Linear Time Sorts
- Stable Sorts and Radix Sort
- Begin Dynamic Programming
- More Dynamic Programming
- Begin Greedy Algorithms: Huffman's Algorithm
- Dÿkstra's Algorithm
- Beyond Asymptotic Analysis: Memory Access Time

- B-Trees
- More B-Trees: Insertion and Splitting
- Union/Find
- Warshall's Algorithm, Floyd's Algorithm
- Large Integer Arithmetic
- RSA Public-Key Cryptosystem
- Begin Algorithms and Structural Complexity Theory
- Continue Algorithms and Structural Complexity Theory
- End Algorithms and Structural Complexity Theory
- Generating Permutations and Combinations
- Exam review with sample questions and solutions.

DATA ANALYSIS AND DESIGN

The choice of a data representation for a problem often affects our thinking about the process. Sometimes the description of a process dictates a particular choice of representation. On other occasions, it is possible and worthwhile to explore alternatives. In any case, we must Analyse and define our data collections.

CONTRACT, PURPOSE, HEADER

We also need a contract, a definition header, and a purpose statement. Since the generative step has no connection to the structure of the data definition, the purpose statement should not only specify what the function does but should also include a comment that explains in general terms how it works.

FUNCTION EXAMPLES

In our previous design recipes, the function examples merely specified which output the function should produce for some given input. For algorithms, examples should illustrate how the algorithm proceeds for some given input. This helps us to design, and readers to understand, the algorithm. For functions such as move-until-out the process is trivial and doesn't need more than a few words.

TEMPLATE

Our discussion suggests a general template for algorithms:

```
(define (generative-recursive-fun problem)
  cond
  [(trivially-solvable? problem)
  (determine-solution problem)]
  [else
  (combine-solutions
  ... problem...
     (generative-recursive-fun (generate-problem-
  1 problem))
       (generative-recursive-fun (generate-                problem-n problem)))]))
```

RANDOMIZED ALGORITHMS

Randomization has become a standard approach in algorithm design and it can be applied in a wide range of problems. Often it leads to more efficient, simpler and shorter solutions than their deterministic counterparts. As opposed to a deterministic algorithm, a randomized algorithm takes a source of random numbers and makes random choices during execution. Hence, the behaviour of the algorithm can change even on a fixed input and is likely to be good on *every* input.

The first class of randomized algorithms are Las Vegas algorithms, which always give *correct* results, the only variation from one run to another being its running time. The running time is a random variable whose expectation is bounded, for example by a polynomial. The second class are Monte Carlo algorithms, whose running time is deterministic, but whose results are only correct with a probability of ≥1/3. A widely known example of a Monte Carlo algorithm is the Miller-Rabin primality test.

QUICKSORT

Quicksort is a divide and conquer sorting algorithm developed by Tony Hoare, who published the first version in the *Communications of ACM*.

The algorithm works by first partitioning (dividing/ rearranging) an input array containing the elements to be sorted into 2 sub-arrays around an element called the pivot *x*, such that all the elements in the lower (left) sub-array are ≤ the pivot *x* ≤ all the elements in the upper (right) sub-array. Following the division is the conquer step, which recursively sorts both sub-arrays. Eventually, there is no work required to combine the sub-arrays since they are sorted in place. A lot of people actually see the recursiveness of Quicksort as one of its weaknesses.

```
void quicksort(int s[], int l, int h)
{
  int p;/* index of partition */
  if ((h - l) > 0) {
  p = partition(s, l, h);
  quicksort(s, l, p - 1);
  quicksort(s, p + 1, h);
  }
```

```
}
int partition(int s[], int l, int h)
{
  int i;
  int p;/* pivot element index */
  int firsthigh;/* divider position for pivot element */
  p = h;
  firsthigh = l;
  for (i = l; i < h; i++)
    if(s[i] < s[p]{
      swap(&s[i], &s[firsthigh]);
      firsthigh++;
    }
  swap(&s[p], &s[firsthigh]);
  return(firsthigh);
}
void swap(int *a, int *b)
{
  int x;
  x = *a;
  *a = *b;
  *b = x;
}
```

Most of people know that Quicksort has an expected running time of $\theta(n \log n)$ on an input array of n elements, which is remarkably efficient. If we are unlucky though, it has worst case running time of $\theta(n^2)$ - which is worse than Heapsort and Mergesort. This behaviour occurs when the partitioning routine produces one sub-array with n–1 elements and another one with 0 elements. That happens when the input array n is already sorted, no matter if forward or backwards. In this situation, even insertion sort would run in the shorter time of *O(n)*. So what can we do about that? What is an easy way to improve the performance of Quicksort? There are actually two ways. One would be to randomize the input array. Another one, the one we will

talk about now, is to pick the pivot randomly.

RANDOMIZED QUICKSORT

The above implementation automatically selects the last element in each sub-array as the pivot. We will now look at a technique called *random sampling*. So instead of always taking the last element in each sub-array, we will select a random element as the pivot during the partitioning. One possible implementation based on our previous code looks like this:

```
int partition(int s[], int l, int h)
{
int i;
int p;/* pivot element index */
int firsthigh;/* divider position for pivot element */
p = l + (random() % (h - l + 1);
swap(&s[p], &s);
firsthigh = l;
for (i = l; i < h; i++)
  if(s[i] < s) {
    swap(&s[i], &s[firsthigh]);
    firsthigh++;
  }
  swap(&s, &s[firsthigh]);
  return(firsthigh);
}
```

Well, so what are the advantages of the randomized version? The running time is independent of input ordering.

- The algorithms does not makes any assumptions about the distribution of the elements in the input array. Already sorted arrays will work as well as random ones.

- No specific input that can produce the worst case behaviour, which is determined only by the random number generator.

As we can see, by introducing some randomness the worst case running time of $\theta(n^2)$ becomes increasingly unlikely and leaves us with the property that the algorithm has a worst-case expected running time of $\theta(n \log n)$. So how to we proof the correctness of this assumption? This is where some probabilistic analysis comes in handy. The following proof is taken from Manuel Blum's lecture notes from his class *CMU 15-451* on algorithms.

ANALYSIS OF RANDOMIZED QUICKSORT

First of all, let's assume that no two elements in the input array are equal. We will be writing will be the quantity we care about (the total number of comparisons) as a sum of simpler random variables, and then just analyze the simpler ones. Define a random variable X_{ij} to be 1 if the algorithm *does* compare the *i*th smallest and *j*th smallest elements during the sort and 0 if it does not. This is called an expectation variable. X denotes the total number of comparisons made by the algorithm. Since we never compare the same pair of elements twice, we have

$$X = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}$$

and therefore

$$E[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E\ X_{ij}$$

Let's consider one of these X_{ij} for $i<j$. Denote the *i*th smallest element by in the array by e_i and the *j*th smallest element by e_j, and conceptually imagine lining up the

elements in sorted order. If the pivot we choose happens to be either e_i or e_j then we compare them. If our pivot is less than e_i or greater than e_j then both end up in the same bucket and we have to choose another pivot. So, we can think of this like a dart game: we throw a dart at random into the array: if we hit e_i or e_j then X_{ij} becomes 1, if we hit between them X_{ij} becomes 0, and otherwise we throw another dart. At each step, the probability that X_{ij} equals 1 conditioned on the event that the game ends in that step is exactly $2/(j-i+1)$. So now we can say that the overall probability that X_{ij} equals 1 is $2/(j-i+1)$.

In other words, for a given element *i*, it is compared to element *i*+1 with probability 1, to element *i*+ 2 with probability 2/3, to element *i*+3 with probability 2/4, to element ..., yes, and so on. That gives us

$$E[X] = \sum_{i=1}^{n} 2\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\ldots+\frac{1}{n-i+1}\right)$$

The quantity $1+1/2+1/3+\ldots+1/n$, denoted H_n, is called the *n*th harmonic number and is in the range $[\ln n, 1+\ln n]$ (this can be seen by considering the integral of $f(x)=1/x$. Therefore,

$$E[X] < 2n(H_n - 1) \le 2n \ln n.$$

SUMMONING THE EVIL

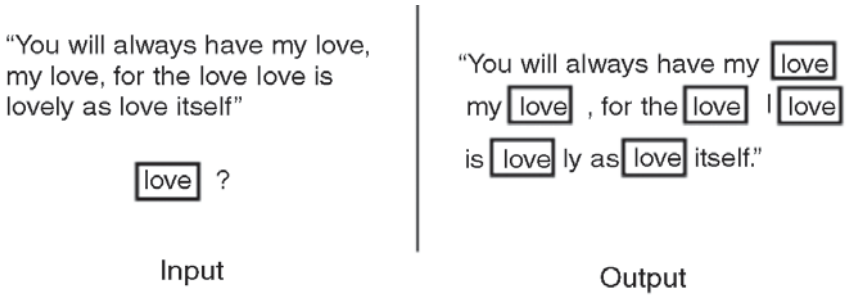
In 1999 Doug McIlroy from *Dartmouth College* published a paper called A Killer Adversary for Quicksort which describes a simple way to find the inputs that even force the randomized Quicksort into the worst case running time of $\theta(n^2)$. It also provides a simple *C* program illustrating the technique.

Quicksort can be made to go quadratic by constructing input on the fly in response to the sequence of items compared. The technique is illustrated by a specific adversary for the standard C qsort function. The general method works against any implementation of quicksort–even a randomizing one–that satisfies certain very mild and realistic assumptions.

CONCLUSION AND ONE MORE THING

This is just a very trivial example of randomized algorithms and many of you are likely to be already familiar with it. Though, if this post happens to attract some interest in this topic, I might write up some more posts about randomized algorithms.

STRING MATCHING



Input Description: A text string t of length n. A patterns string p of length m.

Problem: Find the first (or all) instances of the pattern in the text.

Excerpt from The Algorithm Design Manual: String matching is fundamental to database and text processing applications. Every text editor must contain a mechanism to search the current document for arbitrary strings.

Pattern matching programming languages such as Perl and Awk derive much of their power from their built-in string matching primitives, making it easy to fashion programs that filter and modify text.

Spelling checkers scan an input text for words in the dictionary and reject any strings that do not match.

REGULAR EXPRESSIONS

A regular expression is contained in slashes, and matching occurs with the =~ operator. The following expression is true if the string *the* appears in variable $sentence.

```
$sentence =~/the/
```

The RE is case sensitive, so if

$sentence = "The quick brown fox";

then the above match will be false. The operator !~ is used for spotting a non-match. In the above example

```
$sentence !~/the/
```

is true because the string *the* does not appear in $sentence.

THE $_ SPECIAL VARIABLE

We could use a conditional as

```
if ($sentence =~/under/)
{
   print "We're talking about rugby\n";
}
```

which would print out a message if we had either of the following

```
$sentence = "Up and under";
$sentence = "Best winkles in Sunderland";
```

But it's often much easier if we assign the sentence to

the special variable $_ which is of course a scalar. If we do this then we can avoid using the match and non-match operators and the above can be written simply as

```
if (/under/)
{
   print "We're talking about rugby\n";
}
```

The $_ variable is the default for many Perl operations and tends to be used very heavily.

MORE ON RES

In an RE there are plenty of special characters, and it is these that both give them their power and make them appear very complicated. It's best to build up your use of REs slowly; their creation can be something of an art form.

Here are some special RE characters and their meaning:

```
.  # Any single character except a newline
^     # The beginning of the line or string
$           # The end of the line or string
*     # Zero or more of the last character
+      # One or more of the last character
?       # Zero or one of the last character
```

and here are some example matches. Remember that should be enclosed in/.../slashes to be used.

```
t.e   # t followed by anthing followed by e
# This will match the                    # tre# tle # but not te                                  # tale
^f             # f at the beginning of a line
^ftp         # ftp at the beginning of a line
```

```
e$                  # e at the end of a line
tle$                # tle at the end of a line
und*# un followed by zero or more d characters
# This will match un
                                    # und
                                  # undd
                                # unddd (etc)
.*# Any string without a newline. This is because
# the. matches anything except a newline and
      # the * means zero or more of these.
^$                  # A line with nothing in it.
```

There are even more options. Square brackets are used to match any one of the characters inside them. Inside square brackets a - indicates "between" and a ^ at the beginning means "not":

```
[qjk]    # Either q or j or k
[^qjk]   # Neither q nor j nor k
[a-z]    # Anything from a to z inclusive
[^a-z]   # No lower case letters
[a-zA-Z] # Any letter
[a-z]+   # Any non-zero sequence of lower case letters
```

At this point you can probably skip to the end and do at least most of the exercise. The rest is mostly just for reference.

A vertical bar | represents an "or" and parentheses (...) can be used to group things together:

```
jelly|cream     # Either jelly or cream
(eg|le)gs # Either eggs or legs
(da)+       # Either da or dada or dadada or...
```

Here are some more special characters:

```
\n                                  # A newline
\t                                      # A tab
\w   # Any alphanumeric (word) character.
                  # The same as [a-zA-Z0-9_]
\W                  # Any non-word character.
                 # The same as [^a-zA-Z0-9_]
\d             # Any digit. The same as [0-9]
\D     # Any non-digit. The same as [^0-9]
\s       # Any whitespace character: space,
                              # tab, newline, etc
\S           # Any non-whitespace character
\b       # A word boundary, outside [] only
\B                       # No word boundary
```

Clearly characters like $, |, [,), \,/and so on are peculiar cases in regular expressions. If you want to match for one of those then you have to preceed it by a backslash. So:

```
\|                                  # Vertical bar
\[                  # An open square bracket
\)                    # A closing parenthesis
\*                                  # An asterisk
\^                           # A carat symbol
```

```
\/                              # A slash
\\                              # A backslash
```

and so on.

SOME EXAMPLE RES

As was mentioned earlier, it's probably best to build up your use of regular expressions slowly. Here are a few examples. Remember that to use them for matching they should be put in/.../slashes

```
[01]      # Either "0" or "1"
\/0       # A division by zero: "/0"
\/0       # A division by zero with a space: "/0"
\/\s0     # A division by zero with a whitespace:
          # "/0" where the space may be a tab etc.
\/*0      # A division by zero with possibly some
          # spaces: "/0" or "/0" or "/0" etc.
\/\s*0    # A division by zero with possibly some
          # whitespace.
\/\s*0\.0*      # As the previous one, but with decimal
          # point and maybe some 0s after it. Accepts
          # "/0." and "/0.0" and "/0.00" etc and
          # "/0." and "/0.0" and "/0.00" etc.
```

EXERCISE

Previously your program counted non-empty lines. Alter it so that instead of counting non-empty lines it counts only lines with

- the letter *x*
- the string *the*
- the string *the* which may or may not have a capital *t*
- the word *the* with or without a capital. Use \b to detect word boundaries.

In each case the program should print out every line, but it should only number those specified. Try to use the $_ variable to avoid using the = ~ match operator explicitly.

NP-COMPLETENESS

So far we've seen a lot of good news: such-and-such a problem can be solved quickly (in close to linear time, or at least a time that is some small polynomial function of the input size). NP-completeness is a form of bad news: evidence that many important problems can't be solved quickly.

CARING

These NP-complete problems really come up all the time. Knowing they're hard lets you stop beating your head against a wall trying to solve them, and do something better:

- Use a heuristic. If you can't quickly solve the problem with a good worst case time, maybe you can come up with a method for solving a reasonable fraction of the common cases.
- Solve the problem approximately instead of exactly. A lot of the time it is possible to come up with a provably fast algorithm, that doesn't solve the problem exactly but comes up with a solution you can prove is close to right.

- Use an exponential time solution anyway. If you really have to solve the problem exactly, you can settle down to writing an exponential time algorithm and stop worrying abou t finding a better solution.
- Choose a better abstraction. The NP-complete abstract problem you're trying to solve presumably comes from ignoring some of the seemingly unimportant details of a more complicated real world problem. Perhaps some of those details shouldn't have been ignored, and make the difference between what you can and can't solve.

7

Analysing Animated Cartoons and their Evolution

Cartoons have been and continue to be an important part of our societal culture. Expressive, comedic, and at times political, cartoons appeal to all audiences-both children and adults. “Cartoon” comes from the Italian word “cartone” which means “large paper” (Lobo, 2002). In its simplest form, cartoons are large, pictorial images that serve the purpose of telling a story or commenting on a social or political issue. What makes cartoons so effective despite their simplicity? According to Pulitzer Prize winning author and cartoonist Art Spiegelman, “Comics are the way brains think.

You have small clusters of words in the mind when you speak to someone. These clusters become iconic, abstracting images indicating a visual that becomes real in your brain.”

It is important to know the origin of cartoons and how they have evolved over the centuries to fully appreciate the

cartoons and animations that are produced today. With this knowledge comes the realization that little has changed in the purpose cartoons serve.

The words animation and cartoon have been associated with lively and usually humorous images.

Random House Webster's Unabridged Dictionary defines animation as giving life or liveliness to something and the word cartoon as sketches or drawings similar to the ones we have seen in newspapers. When these words are combined to animated cartoon they refer to a "motion picture consisting of a sequence of drawings, each so slightly different that when filmed and run through a projector the figures seem to move". The animated cartoons discussed in this research refer to the oldest form of animated cartoons, the traditional animation and the latest computerized technology of animation.

Cartoons or animation as we know it today have changed dramatically during the past 100 years. However, animation is much older than that. A recent discovery revealed a 5.200 year old bowl containing a series of related animations of a goat and a fish.

Thousand year old cave paintings and burial chambers which have series of animations that show images can also be related to animation but since, there is no possibility of viewing them in motion they do not qualify as animated cartoons. Whether this sort of animation classifies as an early form of animation is highly debated.

The main aim of this chapter is to inform readers of the history of animated cartoons in relation to their evolution through time and how they have followed latest technologies

throughout the years. The focus is on American and European cartoon animation, since, Asia has their well known Anime cartoons and they need a special research all together and will therefore not be covered here. Animation such as seen in Wallace and Gromit or Robot

Chicken will not be covered either, but they do represent the stop motion technique mentioned later on. The paper will begin by covering the early creators of cartoon animation and early devices. It will then move on to cartoon animation as a business model and take a look at the latest trend and technology in cartoon animation. It ends with the authors conclusions about where cartoon animation is headed in the future.

EARLIEST CARTOONS: EGYPTIANS

The stories and the messages found within cartoons are in reality another means of mass communication. In their most primitive form, cartoons can be found as far back as 1300 BC. In Scott McCloud's *Understanding Comics* , he theorizes that cartoons have been with us since, those ancient times. He points to various murals found along the sides of pyramids and monuments that depict stories. One, for example, describes the daily trials and tribulations that farmers would go through. At the end of the mural, the farmer is beaten to death by one of the pharaoh's tax collectors. It is during this beating that the farmer exclaims, "I hate Mondays!" Here, the Egyptians display their sense of humour and they tell a story through the use of pictures. Therefore, they are implementing the use of what we now consider to be cartoons.

Figure: Example of Egyptian "Cartoon" which Tells about the life of a Farmer.

GREEK ANIMATORS

Egyptians were not the only ones that implemented the use of cartoons in their culture. The Greeks did as well, however theirs are present on their pottery. Greek vases are broken down into two genres: the black figured, and the red figured. The black figured vases from 6th and 5th centuries BC are much more pictorial than the earlier vases. These vases were painted with black figures on reddish orange clay.

The paintings give us a unique understanding into the Greek legends, myths, and the lives of the ordinary people within their society. Examples of stories painted onto the vases are from the Trojan War, the adventures of Odysseus, and those of Dionysos, the Greek god of wine. The red figured vases had a more three dimensional look to them. It is evident here Greek culture tried to gain more depth in their artwork. The characters were usually outlined in black leaving the bare, red clay to show through as the design. This technique also allowed the artist more freedom and therefore the designs on these vases were much more pictorially complicated. It is amazing because from these primitive cartoons we can see how they influenced the foundations of our own society: our government, our literature, and even in our architecture. Here again is a group of images that tell a story and or express social and political ideas. What we now consider to be cartoons existed even back then.

JAPANESE ART

On the other side of the world, the Japanese culture had also developed their own cartoons. These were the first to be on a paper-like substance; they depicted their cartoons on scrolls. Their cartoons told continuous stories. These scrolls first appeared around the 11th and 12th centuries. The "Tale of Genji" picture scrolls are some of the most famous in the Japanese culture. They are also very similar to the ukiyoe prints from the 18th and 19th centuries. In both cases, the figures have the same facial structures and are expressed with simple lines and flat colouring.

In the case of the "Tale of Genji" scrolls, an entire novel is told by the pictures with bare minimal text. This is also one of the earliest examples where pictures are combined with some text to tell a story. Cartoons continue to be an important aspect of the Japanese culture. Their modern cartoons are called "mangas" and are obviously descendents of these early scrolls and ukiyoe prints. They maintain the same line quality and colouring. The Japanese culture is another example of a culture that has stories told by means of pictures as an important part of their history.

Figure: A scroll out of the "Tale of Genji".

EARLY AMERICAN CARTOONS

Along with the development of the printing press, artists and philosophers feared that the new technology would

further sequester men. As a result, they strived to develop a "new language" where pictures and words could be combined; cartoons are acknowledged for the first time at this period. Early artists such as William Hogarth used this new art form as if it were a stage play that incorporates balloons with text. Zincography and photoengraving aided in this new art form's explosion in popularity. Pictures combined with text gave new means to consolidate the advances of all other forms of communication in a cost-effective manner. Thus the birth of cartoons as we know them today.

As time progressed, the printing press, zincography, and photoengraving enabled cartoons to be mass-produced and widely distributed for the first time. This in turn enabled cartoons to be reached by and to affect a larger audience. The printing press reproduced cartoons in black and white and the first political cartoons emerged from its development.

Benjamin Franklin designed one of the first political cartoons in 1754 and to this day it is one of the most famous political cartoons: the "JOIN, or DIE" snake. It discussed the need for the colonies to work together, and while it was most potent during the Revolutionary War, it was originally intended to unite the colonies against the threat of the Indians.

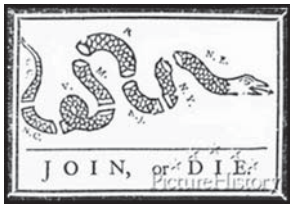


Figure: Benjamin Franklin's "JOIN, or DIE" snake.

20TH CENTURY CARTOONING

Modern cartoons first appeared around the early 1900s. Different types of cartooning began to emerge and they are

classified into the following five groups: illustrative, comic strips, gag strips, animated, and political. Some form of these five different groups appears in most, if not all, newspapers and magazines today. Illustrative cartoons explain stories. They are used in teaching materials and in advertisements. These cartoons have little meaning and are mostly found in schoolbooks.

Comic strips are more often than not found in newspapers and magazines and their purpose is, in essence, to be funny; their intent is to induce laughter from their readers. One of the most famous comic strip artists was Charles Schultz, creator of the *Peanuts* comic strip.

Gag strips are usually composed of a single picture combined with one to two sentences and also have the job of producing laughter. Animated cartoons are the latest of the cartoon form. They can be done by both hand and by computer and appeal to children as well as to adults. The main characteristic of animated cartoons that differentiates them from the other types is that animated cartoons also involve the medium of movement.

Political cartoons are intended for adults and usually convey a point of view concerning a societal issue current to its time of publication. These also appear in newspapers and magazines.

WALT DISNEY

Cartoons did not evolve on their own; various influential people made their own lasting mark on this art and communication form. The first and foremost important person from the field of animated cartoons is Walt Disney. His first cartoon character was named Oswald Rabbit. This

was his first comic strip as well as his first cartoon series. He created this character as well as the strip along with his brother. Following the creation and success of Oswald Rabbit came Steamboat Willie, one of the most famous cartoon characters. Steamboat Willie led the way for Mickey Mouse, Minnie Mouse, Donald, Daisy, and the other Disney characters. Disney created the first full-length animated film back in 1937 with *Snow White and the Seven Dwarfs.* This was also Disney's first attempt at a musical, and it was a tremendous success. The success of this film spawned the creation of many other animated, musical, and feature films.

Figure: Scene out of Disney's *Snow White.*

CHUCK JONES

Another influential person in the field of cartoons is Chuck Jones. While his name may not be as well known as Disney, his characters make up for it. Like Disney, Jones created characters for predominantly animated cartoons. In 1932, Jones was a cell washer at Ubbe Iwerks Studios and he later joined the Leon Schliesinger Studios.

Not long after, Warner Brothers bought these studios. The characters he created there are part of what put Warner Brothers on the map. These characters are an integral part

of our society: Bugs Bunny, Daffy Duck, Elmer Fudd, and Porkey Pig. Jones not only created the characters but also did their animations as well as their voices and personas. However Jones did not solely work for Warner Brothers, he also created some characters on his own. These characters may also seem familiar: Road Runner, Wile E. Coyote, Marvin the Martian, Pepe le Pew, and Gossemer. Jones may not be as well known as others like Disney, however his impact upon the cartooning community is just as immeasurable.

Figure: Bugs Bunny with many other Warner Brothers cartoons on the bottom.

HANNA-BARBERA

Another group that made their mark on cartooning is William Hanna and Joseph Barbera; this duo is better known as Hanna-Barbera. From 1960 to 1966, Hanna-Barbera's cartoons "The Flintstones" and "Atom Ant" were broadcast. Their success led to the creation of other memorable cartoons such as "Yogi Bear", "Johnny Quest", "The Jetsons", "Tom and Jerry", and the very memorable "Scooby Doo." There are many others that joined the Hanna-Barbera cartoon family; these are just a few examples. For over 60 years these two cartoonists and animators shaped what we think of when

we hear, "Saturday morning cartoons." William Denby Hanna was born in Melrose, New Mexico, in 1910. Soon after, his family moved to Los Angeles. There he was an active member of the Boy Scouts. Joseph Roland Barbera was born in 1911 in the Little Italy section of New York's Lower East Side, but was raised in Brooklyn. In 1937, Bill Hanna and Joe Barbera joined forces. Coming from different backgrounds and experiences, and possessing different talents, they were worried about their compatibility. However the gamble was a success and they completely complimented each other.

Figure : Tom and Jerry (1999).

Figure : Scooby Doo (2000).

Since, the masterminds of animated cartoons have been acknowledged, the comic strip area of cartooning must not be forgotten. As previously mentioned, Charles Schultz is the Walt Disney of this type of cartooning. From when he was little, Schultz knew that he wanted a job as a cartoonist. After returning from the war, Schultz began his working career lettering tombstones in St. Paul, Minnesota, however

this did not last long. From there he took a job at a Roman Catholic magazine called *Timeless Topix.* He did work there – both pictorial and written. Working for *Timeless Topix* was his first real job in cartooning. He took a second job as a teacher to pay the bills and it was as a teacher that he met many of the people he would later use as models for characters in the *Peanuts* strip. Then, on October 2, 1950, the world was introduced to some characters that would change their lives forever. During the time when America was still rejoicing from the war and no person was supposed to be unhappy, Schulz introduced us to Charlie Brown, Snoopy, Lucy, and the rest of the Peanuts gang. At our first introduction, we were introduced to the "L'il Folks", but we came to know them as the Peanuts crew. His work was new and different. Everyone could identify with main character who had the same problems as they did. The only difference was that that character acknowledged his problems. Charlie Brown represented the inner psyche of the American people. Schultz had the pulse of the every American in his hand and to this day the *Peanuts* strip is the most widely syndicated strip ever. It has appeared in over two thousand six hundred different newspapers in seventy-five countries and in twenty-one different languages. Charles Schultz and his most popular character, Charlie Brown, will forever impact the American people and the way they view themselves.

MODERN MASTERS

As times change, cartoons and the way in which they are made change as well. Back in 1995, Disney joined up with Pixar to create *Toy Story,* the first full-length feature film created entirely on a computer. *Toy Story*'s success led to

the production of other computer-animated movies such as *A Bug's Life* in 1998, *Toy Story 2* in 1999, and more recently, *Monster's INC.* in 2001. It is particularly true of these cartoons that they appeal to all audiences. Some of the dialogue and situations are so that only an adult would be able to fully appreciate the humour and wit behind them. On the other hand, the bright and playful colours, the characters, and the basic story lines are for the children. Either way, this new type of cartoon continues to entertain any and every person.

Since, these films are made completely by a computer, the production aspects of these films vary from the traditional animated films. The animators at Pixar, in creating the films, neither draw nor paint each successive scene like done in traditional animations. Using Pixar's animation software, the animators choreograph the movements and facial expressions in each of the scenes and then computers generates the "in between" frames, which are adjusted as needed. Currently, computer animation in the cartoon genre is at its peak.

COMPUTER GAMES

Computer gaming is a vibrant multibillion dollar industry that offers exciting career opportunities for computer scientists as well as visual artists.

Computer games in the 21st century are a form of entertainment that rivals listening to music and watching movies. In fact, according to a study from research firm NPD Group, published in 2008, 72 percent of the U.S. population actively play games. Computer games can vary wildly in their themes and style, but at a basic level involve manipulating

events on-screen via a control input device, such as a joypad or keyboard, in order to accomplish in-game objectives.

HISTORY

The evolution of computer gaming can be traced back to tic-tac-toe simulators on 1940s cathode ray tube devices, as well as very basic titles played on mainframe machines. Gaming truly exploded into the public domain, though, with the advent of coin-op machines in arcades, which began to appear in the early 1970s. Arcade titles such as "Pong" by Atari and "Death Race" were popular. The first home gaming console, the Magnavox Odyssey, appeared in 1972. From there, both console and PC gaming began to escalate in terms of both popularity and power, with big names such as Microsoft, Sony, Nintendo and Sega all releasing home consoles. By 2011, PCs and gaming consoles support titles that feature vast worlds and highly advanced 3D graphics.

GAME FORMATS

Video gaming can be divided roughly into two spheres: PC gaming, involving personal computers such as laptop devices, and console gaming. PC gaming utilizes a computer's mouse and keyboard to direct the action on-screen, and includes everything from Flash-technology games played over the Internet to complex titles purchased from stores. Gaming consoles utilize video display signals, with the in-game action taking place on a TV screen. Twenty-first-century consoles allow access to the Internet, a feature not found on earlier machines. Portable devices such as the Nintendo DS as well as cell phones, which contain a small screen and compact controls, allow for handheld gaming.

GENRES

Among games, several genres have achieved popularity and continue to influence new titles. These include shooter games, which find players blasting enemies on the screen, and often use a first-person perspective, as seen in titles such as "Quake" and "Halo." Platform games, such as the "Super Mario" series, see players directing characters across 2D or 3D environments, with precise timing used to avoid enemies and obstacles. In role-playing games, such as "Fallout," players create a character and take part in quests, often developing their character as they progress. Sports titles simulate real-life activities, and include everything from Formula One to professional wrestling games.

MULTIPLAYER

Many games allow more than one player to participate, typically simultaneously. In the 21st century, multiplayer gaming usually takes place via the Internet, with players all in their own home, with their own copies of the game. Popular Internet games include "World of Warcraft," a role-playing title enjoyed by thousands of players who interact online. Gaming consoles allow two or more players to participate at once, however, by using multiple control devices attached to one machine.

GAME PROGRAMMING

Game programming, a subset of game development, is the software development of video games. Though often engaged in by professional game programmers, many novices may program games as a hobby. Some software engineering

students program games as exercises for learning a programming language or operating system.

DEVELOPMENT PROCESS

Professional game development usually begins with a game design, which itself has several possible origins. Occasionally the game development process starts with no clear design in mind, but as a series of experimentation. For example, game designer Will Wright began development of *The Sims* by getting programmers to experiment with several ideas.

PROTOTYPING

Programmers are often required to produce prototypes of game play ideas and features. A great deal of prototyping may take place during pre-production, before the design document is complete, and may help determine what features the design specifies. Prototypes are developed quickly with very little time for up-front design and mostly act as a proof of concept or to test ideas.

GAME DESIGN

Though the programmer's main job is not to develop the game design, the programmers often contribute to the design, as do game artists. The game designer will solicit input from both the producer and the art and programming lead for ideas and strategies for the game design. Often individuals in non-lead positions also contribute, such as copywriters and other programmers and artists.

Programmers often closely follow the game design document. As the game development progresses, the design document changes as programming limitations and new capabilities are discovered and exploited.

PRODUCTION

During production, programmers churn out a great deal of source code to create the game described in the game's design document. Along the way, the design document is modified to meet limitations or expanded to exploit new features. The design document is very much a "living document" much of whose life is dictated by programmer's schedules, talent and resourcefulness.

While many programmers have some say in a game's content, most game producers solicit input from the lead programmer as to the status of a game programming development. The lead is responsible for knowing the status of all facets of the game's programming and for pointing out limitations. The lead programmer may also pass on suggestions from the programmers as to possible features they'd like to implement. With today's visually rich content, the programmer must often interact with the art staff. This very much depends on the programmer's role, of course. For example, a 3D graphics programmer may need to work side by side with the game's 3D modelers discussing strategies and design considerations, while an AI programmer may need to interact very little, if at all, with the art staff. To help artists and level designers with their tasks, programmers may volunteer or be called upon to develop tools and utilities. Many of these may be *ad-hoc* and buggy due to time constraints (time for development of such tools is often not included in a game's schedule) as well as because they are only for in-house use anyway. Many game tools are developed in RAD languages for quicker development and may be discarded after the completion of the game.

TESTING

The formal quality assurance testing process, performed by professional game testers, begins well into game development. High-budget titles may begin testing with the first playable alpha, while low-budget and casual games might not enter testing until a release candidate is ready. The programmers' task is to fix errors and bugs as such are discovered by the QA teams.

NEARING COMPLETION

Final tasks include "polishing" the game, such as programmers fixing occasional bugs—from minor to catastrophic—that may arise during the last phases of testing. Game developers may have a beta testing period, but the definition of such varies from developer to developer. Often a beta contains all of the game's features, but may have a few bugs or incomplete content. Few games are given a public beta period, for example, to measure stress tolerance for game servers. When the game is deemed complete, it is said to have "gone gold" and is shipped off to the publisher. Depending on circumstances, the publisher may then subject it to its own quality assurance or may begin pressing the game from the gold master.

MAINTENANCE

Once a game ships, the maintenance phase for the video game begins. Programmers wait for a period to get as many bug reports as possible. Once the developer thinks they've obtained enough feedback, the programmers start working on a patch. The patch may take weeks or months to develop, but it's intended to fix most bugs and problems with the

game. Occasionally a patch may include extra features or content or may even alter gameplay.

DURATION

Most modern games take from one to three years to complete. The length of development depends on a number of factors, but programming is required throughout all phases of development except the very early stages of game design.

TOOLS

Game development programs are generated from source code to the actual program (called the *executable*) by a compiler. Source code can be generated by almost any text editor, but most professional game programmers use a full Integrated Development Environment (IDE). Once again, which IDE one uses depends on the target platform. Popular ones for Xbox and Windows development are Microsoft Visual Studio and CodeWarrior.

In addition to IDEs, many game development companies create custom tools developed to be used in-house. Some of these include prototypes and asset conversion tools (programs that change artwork, for example, into the game's custom format). Some custom tools may even be delivered with the game, such as a level editor.

Game development companies are often very willing to spend thousands of dollars to make sure their programmers are well equipped with the best tools. A well outfitted programmer may have two to three development systems dominating their office or cubicle.

Once the game's initial design has been agreed upon, the development language must be decided upon. The choice

depends upon many factors, such as language familiarity of the programming staff, target platforms (such as PlayStation or Microsoft Windows), the execution speed requirements and the language of any game engines, APIs or libraries being used.

Table: Programming Languages

Language	***Strengths***	***Weaknesses***
Assembly	Low overhead	Error-prone, slow development, difficult for novices, not portable
C	Widely known, numerous tools	No built-in OO support, difficult for large projects or multiple platforms.
C++	Built-in OO support, widely used, numerous tools	No protection from memory leaks
C#	Very OO, RAD language, easy to use	High memory usage
Java	Very OO, easy to use portable	Not generally available on game consoles
Eiffel, Smalltalk, Ada, etc.		Fringe game languages, few game development tools
Scripting languages like Lua, Python, etc.		Often used for gameplay scripting, but not for the bulk of the game code itself

Today, because it is object oriented and compiles to binary (the native language of the target platform), the most popular game development language is C++. However, Java and C are also popular, but inappropriate for some projects. Assembly language is necessary for some video game console programming and in some routines that need to be as fast as possible, or require very little overhead. C# is popular for developing game development tools.

High-level scripting languages are increasingly being used as embedded extensions to the underlying game written in a low or mid-level programming language such as C++. Many developers have created custom languages for their games, such as id Software's QuakeC and Epic Games' UnrealScript. Others have chosen existing ones like Lua

and Python in order to avoid the difficulties of creating a language from scratch and teaching other programmers a proprietary language.

Vertex and pixel shaders are increasingly used in game development as programmable GPUs have become more prevalent. This has led to the increasing use of High Level Shader Languages in game programming, such as nVidia's Cg, though it cannot be used for all of game logic.

APIS AND LIBRARIES

A key decision in game programming is which, if any, APIs and libraries to use. Today, there are numerous libraries available which take care of key tasks of game programming. Some libraries can handle sound processing, input, and graphics rendering. Some can even handle some AI tasks such as pathfinding. There are even entire game engines that handle most of the tasks of game programming and only require coding game logic.

Which APIs and libraries one chooses depends largely on the target platform. For example, libraries for PlayStation 2 development are not available for Microsoft Windows and vice-versa. However, there are game frameworks available that allow or ease cross-platform development, so programmers can program a game in a single language and have the game run on several platforms, such as the Wii, PlayStation 3, Xbox 360, Xbox, PSP and Microsoft Windows.

GRAPHIC APIS

Today, graphics are a key defining feature of most games. While 2D graphics used to be the norm for games released through the mid-1990s, most games now boast full 3D

graphics. This is true even for games which are largely 2D in nature, such as *Civilization III*.

The most popular personal computer target platform is Microsoft Windows. Since it comes pre-installed on almost ninety percent of PCs sold, it has an extremely large user base. The two most popular 3D graphics APIs for Microsoft Windows are Direct3D and OpenGL. The benefits and weaknesses of each API are hotly debated among Windows game programmers. Both are natively supported on most modern 3D hardware for the PC.

DirectX is a collection of game APIs. Direct3D is DirectX's 3D API. Direct3D is freely available from Microsoft, as are the rest of the DirectX APIs. Microsoft developed DirectX for game programmers and continues to add features to the API. The DirectX specification is not controlled by an open arbitration committee and Microsoft is free to add, remove or change features. Direct3D is not portable; it is designed specifically for Microsoft Windows and no other platform (though a form of Direct3D is used on Microsoft's Xbox, Windows Phone 7.5 smartphones and mobile devices which run the Pocket PC operating system). The DirectX API is updated far more often than OpenGL implementations. As a result, new features of the latest 3D cards are included in the API much faster than with OpenGL.

OpenGL is a portable API specification. Code written with OpenGL is easily ported between platforms with a compatible implementation. *Quake II* was ported from Windows to Linux by a fan of the game. OpenGL is a standard maintained by the OpenGL Architecture Review Board (ARB). The ARB meets periodically to update the standard by adding emerging

support for features of the latest 3D hardware. Since it is standards based and has been around the longest, OpenGL is used by and taught in colleges and universities around the world. In addition, the development tools provided by the manufacturers of some video game consoles (such as the Nintendo GameCube, the Nintendo DS, and the PSP) use graphic APIs that resemble OpenGL.

OpenGL often lags behind on feature updates due to the lack of a permanent development team and the requirement that implementations begin development after the standard has been published. Programmers who choose to use it can access some hardware's latest 3D features, but only through non-standardized extensions. The situation may change in the future as the OpenGL architecture review board (ARB) has passed control of the specification to the Khronos Group in an attempt to counter the problem.

OTHER APIS

For development on Microsoft Windows, the various APIs of DirectX may be used for input, sound effects, music, networking and the playback of videos. Many commercial libraries are available to accomplish these tasks, but since DirectX is available for free, it is the most widely used.

For console programming, the console manufacturers provide facilities for rendering graphics and the other tasks of game development. The console manufacturers also provide complete development systems, without which one cannot legally market nor develop games for their system. Third-party developers also sell toolkits or libraries that ease the development on one or more of these tasks or provide special benefits, such as cross-platform development capabilities.

GAME STRUCTURE

The central component of any game, from a programming standpoint, is the *game loop*. The game loop allows the game to run smoothly regardless of a user's input or lack thereof.

Most traditional software programs respond to user input and do nothing without it. For example, a word processor formats words and text as a user types. If the user doesn't type anything, the word processor does nothing. Some functions may take a long time to complete, but all are initiated by a user telling the program to do something.

Games, on the other hand, must continue to operate *regardless* of a user's input. The game loop allows this. A highly simplified game loop, in pseudocode, might look something like this:

```
while( user doesn't exit )
 check for user input
  run AI
  move enemies
  resolve collisions
  draw graphics
  play sounds
end while
```

The game loop may be refined and modified as game development progresses, but most games are based on this basic idea.

Game loops differ depending on the platform they are developed for. For example, games written for DOS and most consoles can dominate and exploit available processing resources without restraint. However, game for a modern PC operating system such as Microsoft Windows must operate within the constraints of the process scheduler. Some modern games run multiple threads so that, for example, the computation of character AI can be decoupled from the

generation of smooth motion within the game. This has the disadvantage of (slightly) increased overhead, but the game may run more smoothly and efficiently on hyper-threading or multicore processors and on multiprocessor platforms. With the computer industry's focus on CPUs with more cores that can execute more threads, this is becoming increasingly important. Consoles like the Xbox 360 and PlayStation 3 already have more than one core per processor, and execute more than one thread per core.

HOBBYISTS

The only platforms widely available for hobbyists to program are consumer operating systems. This is because development on game consoles requires special development systems that cost thousands of dollars. Often these must be obtained from the console manufacturer and are only sold or leased to professional game development studios. However, Microsoft distributes a game development framework, XNA, which runs on both Microsoft Windows and Xbox 360. Games written for Windows often can be ported to Xbox with few changes. This allows individuals and small teams to develop games for consoles. Some hobbyists also develop homebrew games, especially for handheld systems or obsolete consoles.

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.