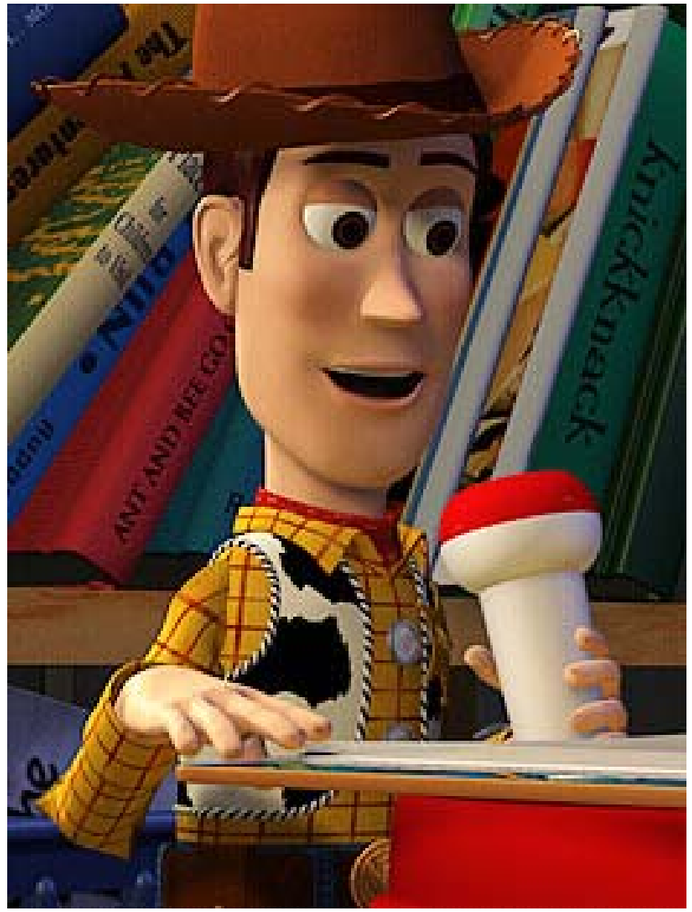
Computer Graphics

Comunicación Visual. Tecnun

Nicolás Serrano – 2019. Tecnun

Introduction

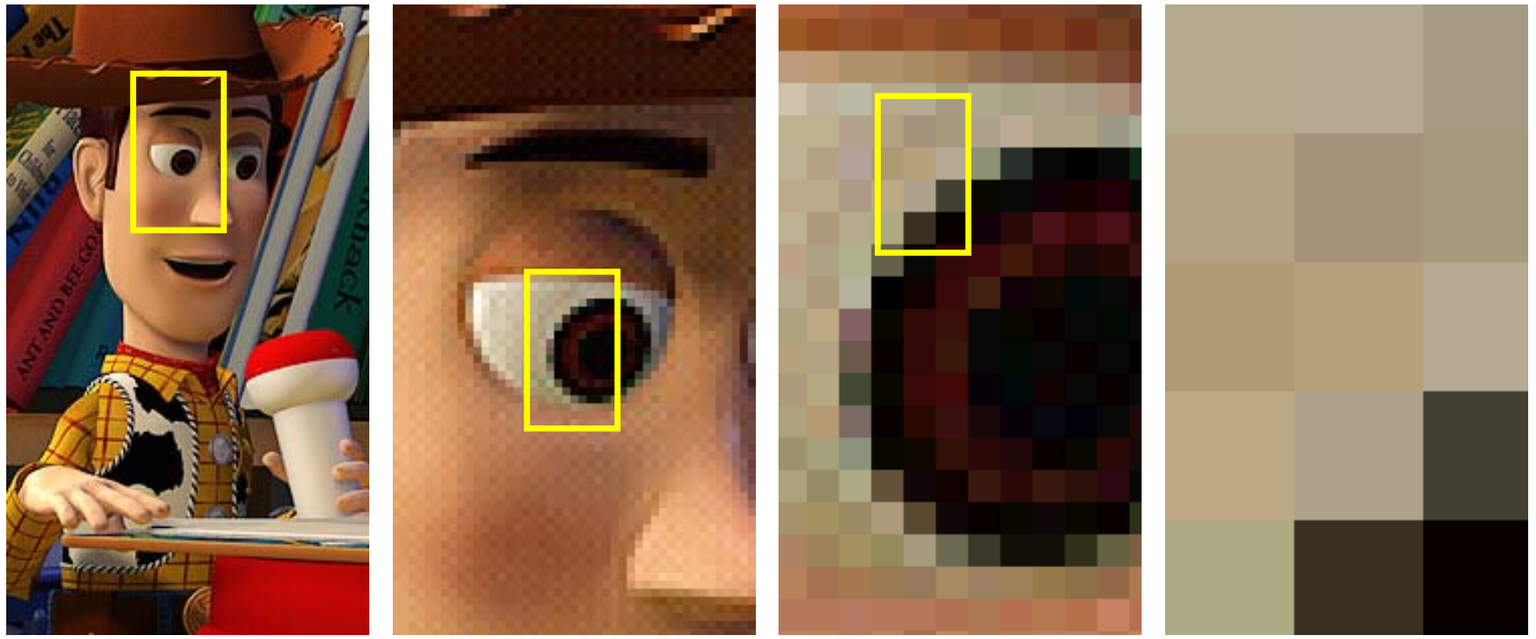


Agenda

- Introduction to Computer Graphics
- History
- CG in films
- Applications
- Contents of the course

What are Computer Graphics?

- Edition of models and figures
  - Create, store, modify and render
- The final output on the screen are **pixels**



Elements of Computer Graphics

- Hardware
  - CPU, video card, display, ...
- Software
  - modeler, capturer, renderer, …
- People
  - programmer, designer, animator, …
- Products
  - applications, films, images, models, ...

Short history of Computer Graphics

- 1940: first project of a Computer Graphics system
- 1951: Whirlwind: representation of data of a radar
- 1961: Spacewar, first video game

Sketchpad

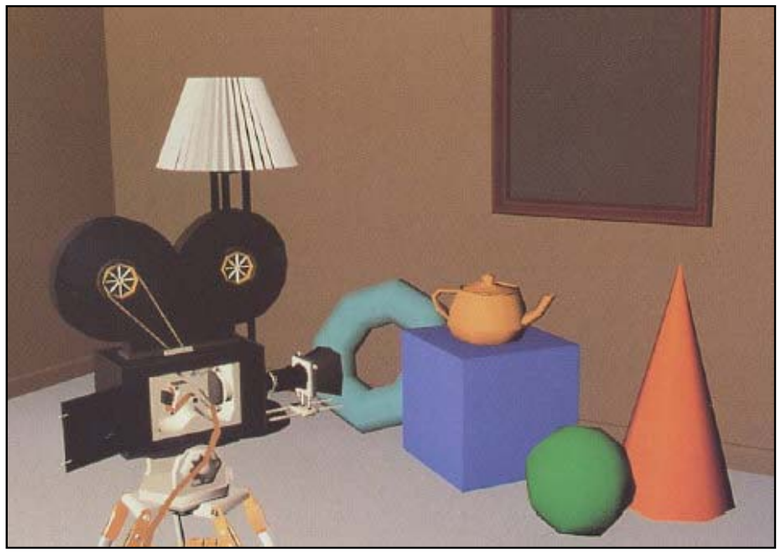
- Ivan Sutherland (doctoral thesis in MIT, 1963)
- First interactive graphic system
  - Display of primitives
  - Hierarchic models
  - Constraints based
  - Optic pen
  - Function keys
  - Popup menus

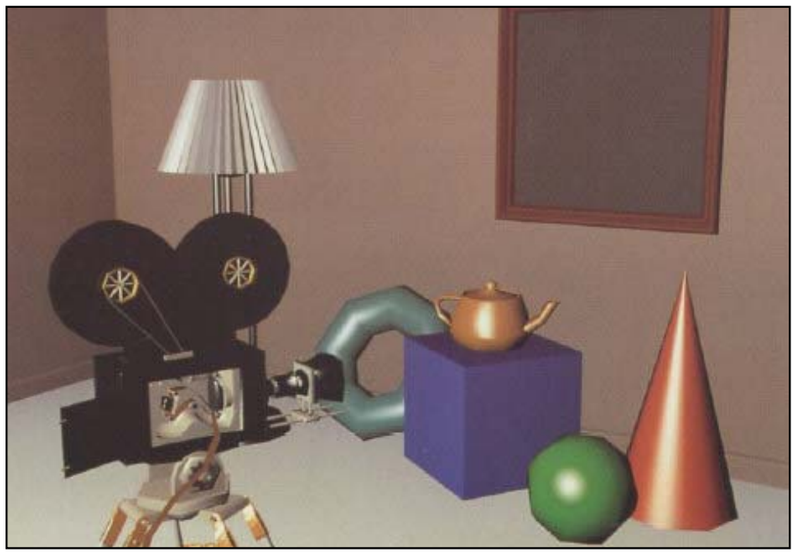
Evolution of hardware

- Vectorial displays
  - based on oscilloscopes
- 1975: Cathode Ray Tube (CRT)
  - Time independent of number of elements
  - introduction of frame buffer
- 1980: low cost hardware -> spreading of graphic applications

Rendering

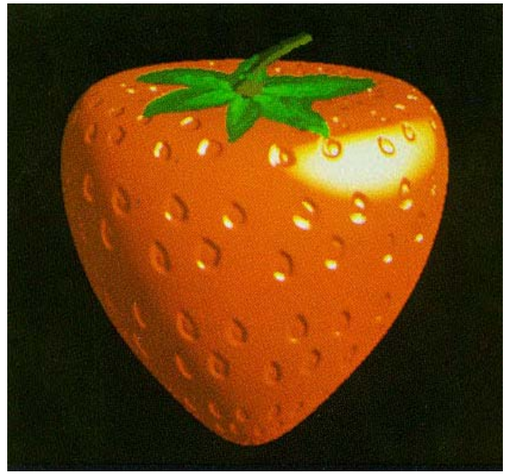
- 1971 – Gouraud surface rendering
- 1974 - Z-Buffer algorithm, Catmull
- 1975 – Phong surface rendering
- 1975 – Fractal geometry of Mandelbrot

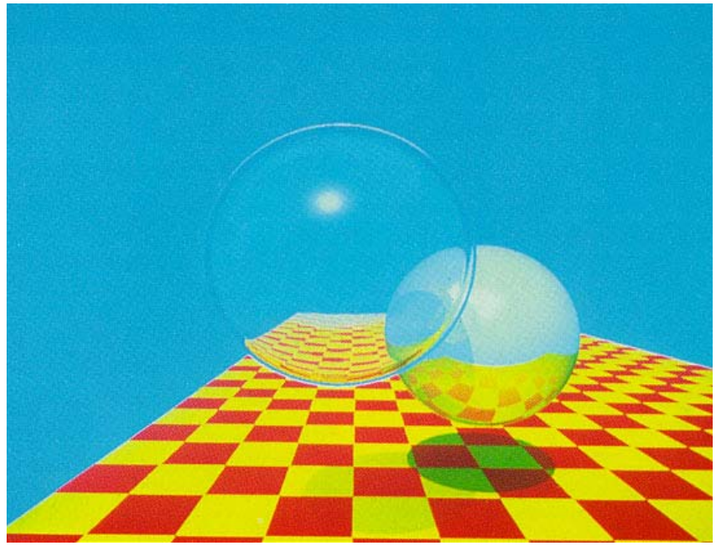




Rendering

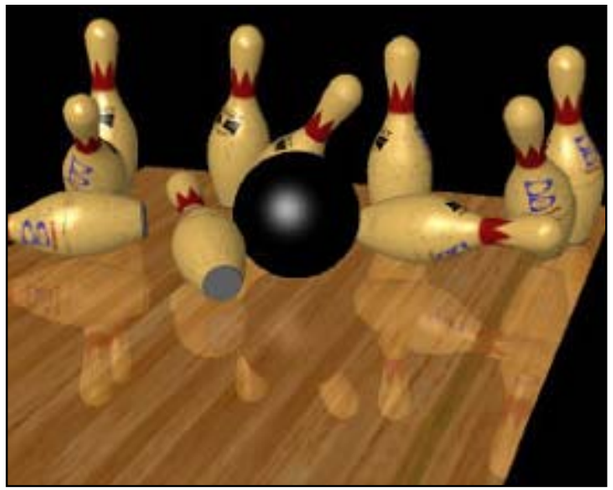
- 1976 - Textures, James Blinn, Catmull
- 1978 – Curve surfaces, James Blinn
- 1980 - Ray tracing, Turner Whitted
- 1984 - Radiosity, Cornell University





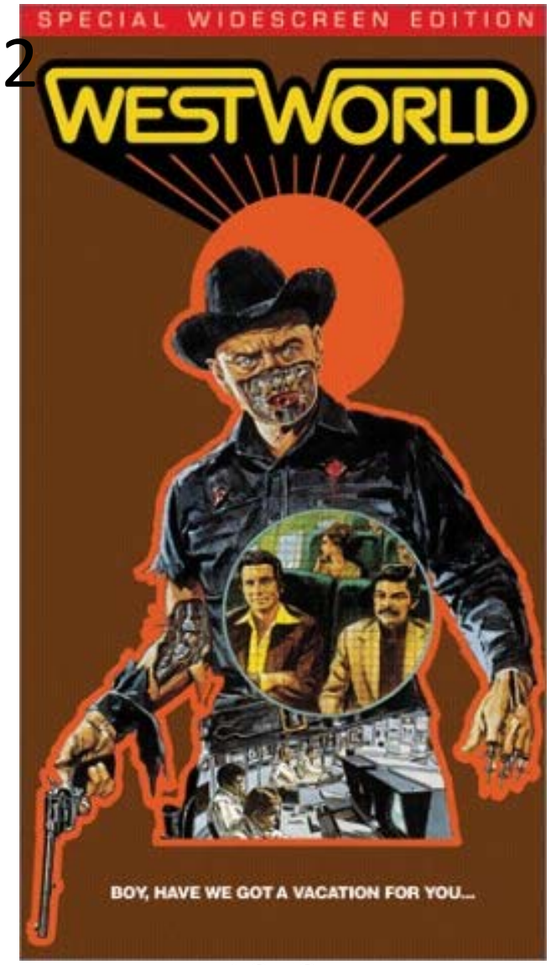
Rendering

- 1985 – Solid textures, Perlin
- 1988 - Renderman, Pixar
- 1995 – first feature film:
  Toy Story

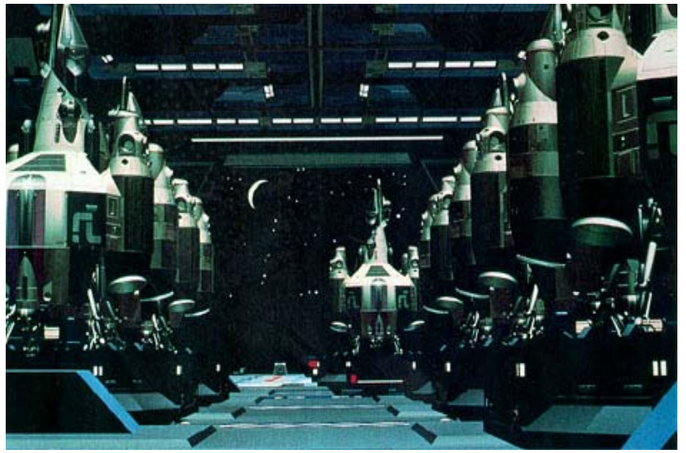


Computer Graphcis in films

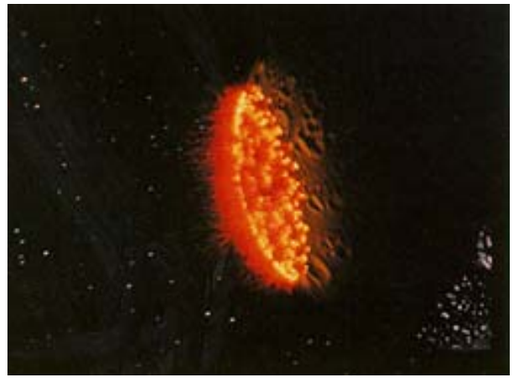
- 1982



1973

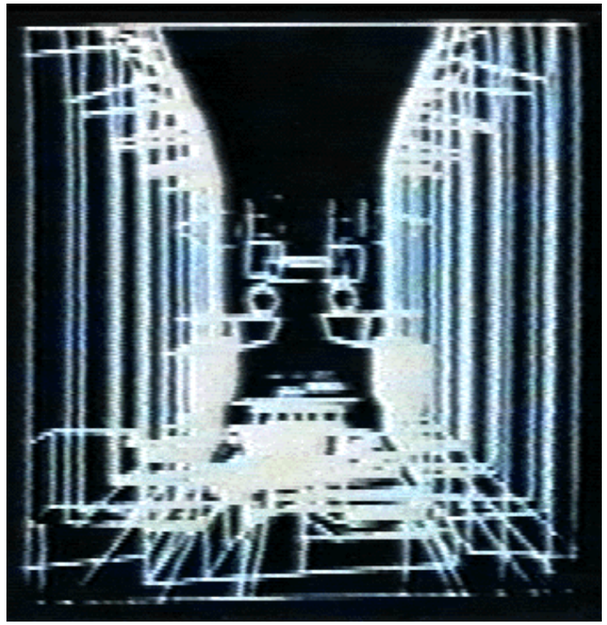


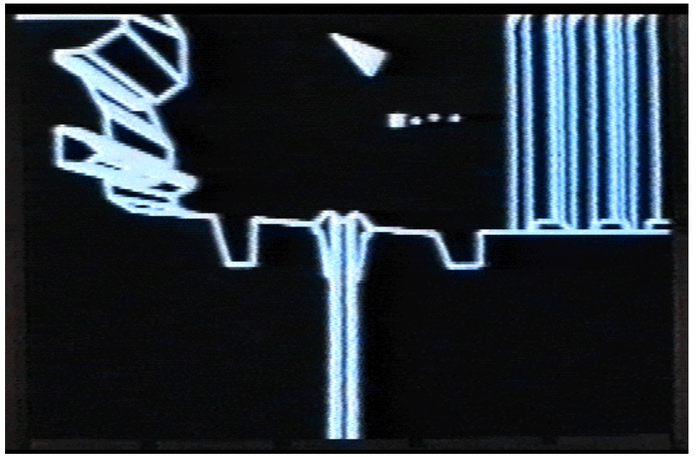
1982



Star Trek II - 1983

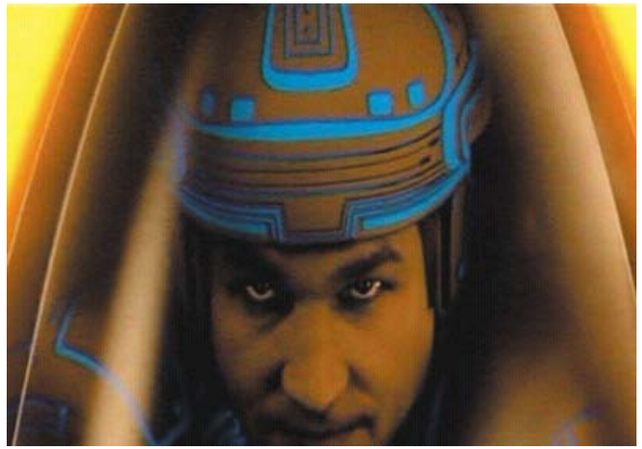
Star Wars

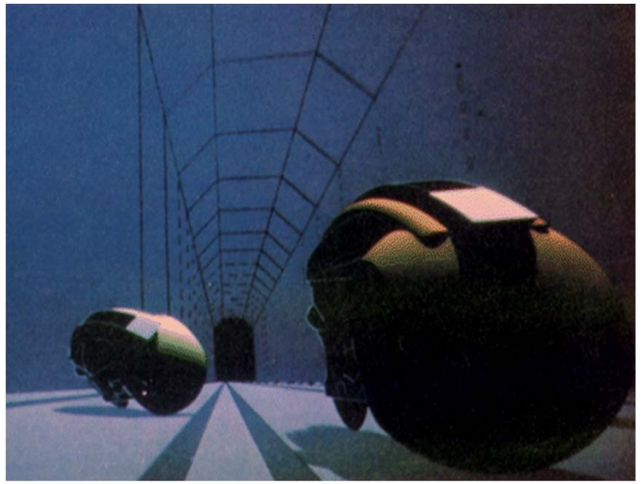


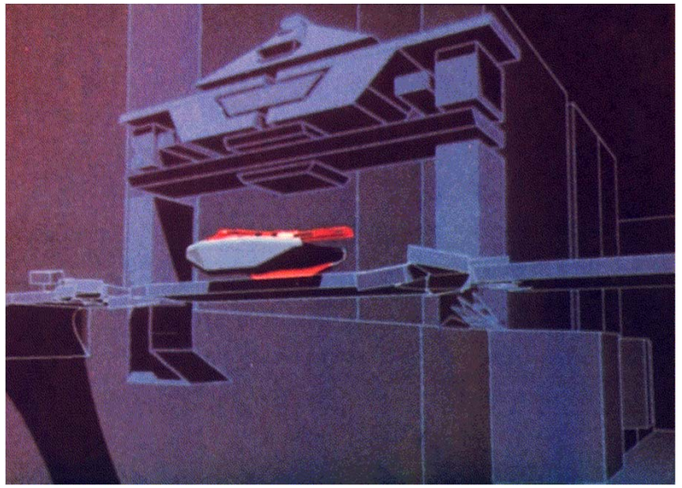


1977

Tron







1980

Feature films made with computer graphics

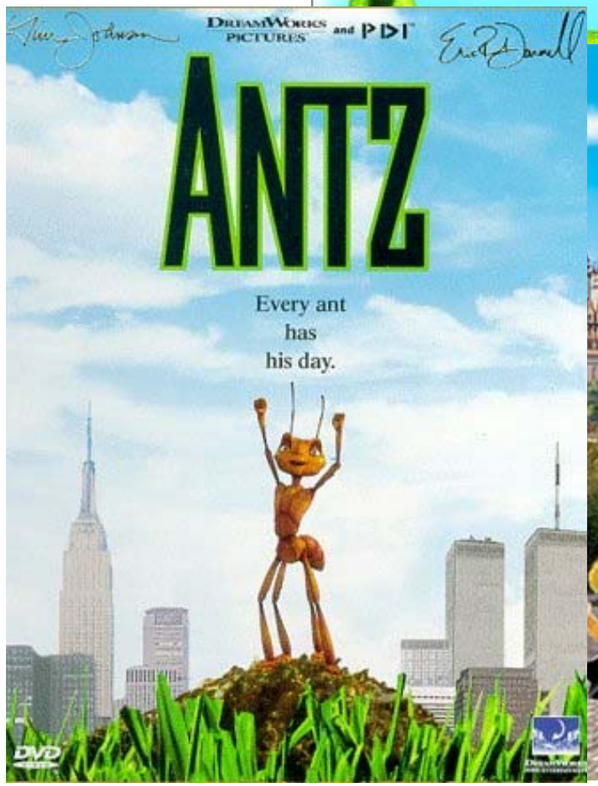
- Toy Story
- Bugs (Bichos)
- Hormigaz (Antz)
- Monsters Inc.
- Shrek
- Toy Story 2
- Dinosaurs
- Ice Age, Barbie, …
- Finding Nemo

Pixar


TOY STORY
a bug's life
TOM HANKS TIM ALLEN
Disney·PIXAR
TOY STORY 2
From the Creators of TOY STORY
MONSTERS, INC.
FINDING NEMO
THE INCREDIBLES
CARS
June 9 2006

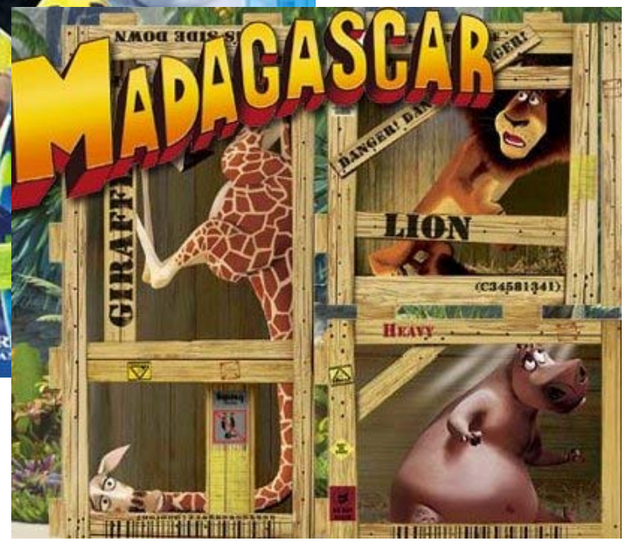
orks


MYERS MURPHY DIAZ LITHGOW
SHREK


DREAMWORKS PICTURES and PDI
ANTZ
Every ant has his day.


DREAMWORKS
SHREK 2


Will Smith
Robert De Niro
Renée Zellweger
Angelina Jolie
Jack Black
Martin Scorsese
Shark Tale
FROM THE STUDIO THAT BROUGHT YOU SHREK

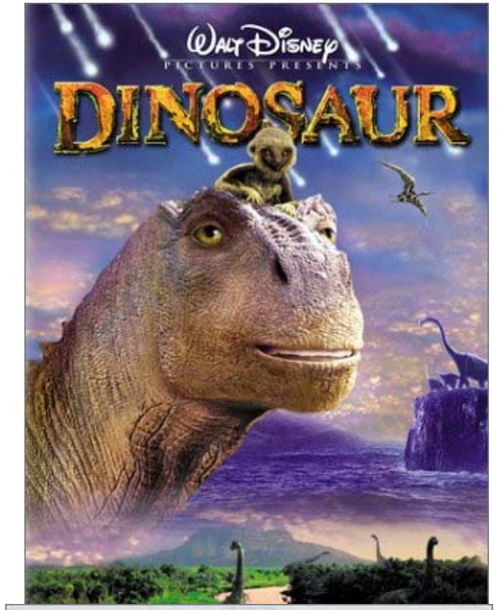

MADAGASCAR
GIRAFFE
LION
HEAVY

S

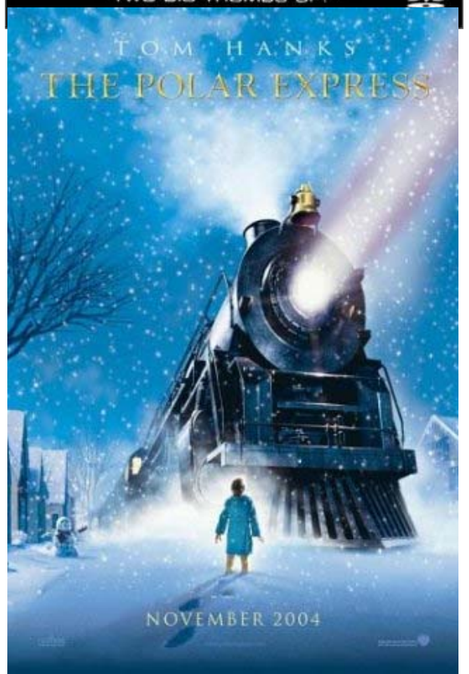

2-DISC SPECIAL EDITION
ICE AGE
SCRAT'S MISSING

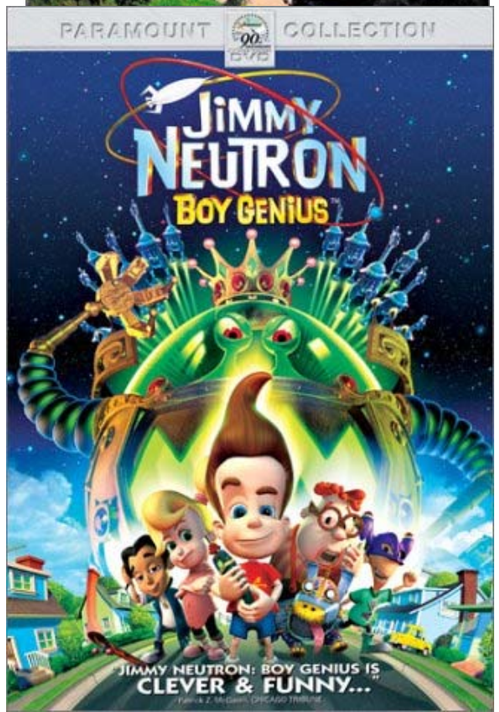

DE LOS CREADORES DE ICE AGE
ROBOTS
MARZO 2005


SPECIAL EDITION
FINAL FANTASY
THE SPIRITS WITHIN
UNLEASH A NEW REALITY
"TWO BIG THUMBS UP!"
DVD


WALT DISNEY
PICTURES PRESENTS
DINOSAUR


STARRING in Her FIRST MOVIE
Barbie
STARRING in a FAIRY TALE ADVENTURE
Barbie
DISCOVER A MAGICAL NEW LAND
Barbie
Fairytopia
ENTER A WORLD OF LOVE AND
Barbie
of Swan Lake
SINGING IN HER FIRST MOVIE MUSICAL!
Barbie
as The Princess and the Pauper


TOM HANKS
THE POLAR EXPRESS
NOVEMBER 2004

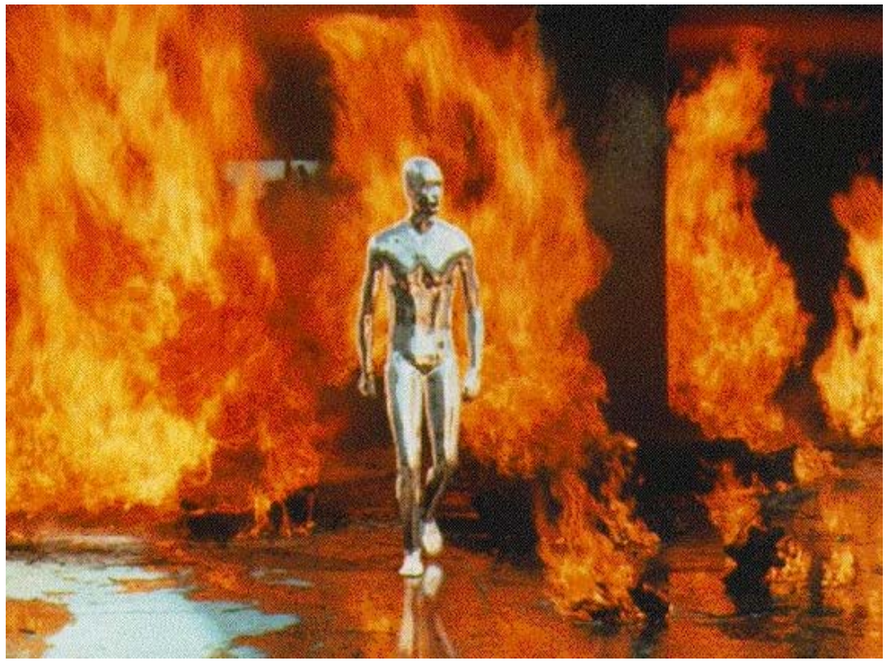

PARAMOUNT
COLLECTION
JIMMY NEUTRON
BOY GENIUS
JIMMY NEUTRON: BOY GENIUS IS CLEVER & FUNNY...

Simulation of “reality”

- Stars Wars, Episode 1
- Jurassic Park
- Final Fantasy

Capture of real movement

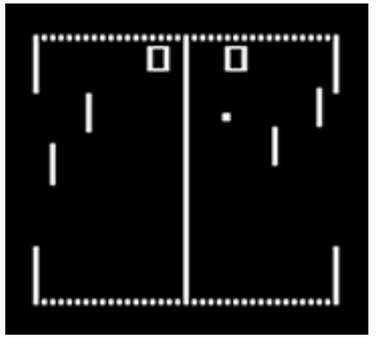
Special effects

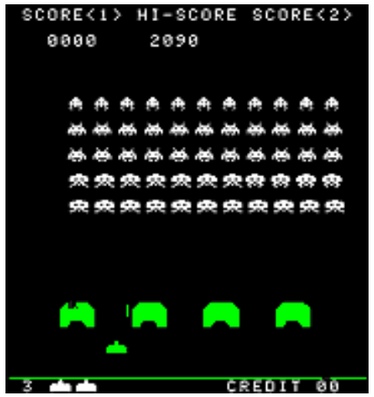


The Adventures of Rocky & Bullwinkle, The Beach, Cast Away, Chicken Run, Dinosaur, Gladiator, Hollow Man, How the Grinch Stole Christmas, Mission to Mars, The Perfect Storm, Pitch Black, Nutty Professor II: The Klumps, Space Cowboys, Bicentennial Man, Fight Club, Inspector Gadget, Iron Giant, The Matrix, Sleepy Hollow, Star Wars: The Phantom Menace, Stuart Little, Toy Story 2, The World is Not Enough, Babe: Pig in the City, A Bug's Life, Patch Adams, What Dreams May Come, Contact, Jurassic Park: The Lost World, Men in Black, Starship Troopers, Apollo 13, Balto, Batman Forever, Casper, Indian in the Cupboard, Jumanji, Outbreak, Pocahontas, Species, Toy Story, Clear & Present Danger, Forrest Gump, Interview with a Vampire, The Jungle Book, The Lion King, The Mask, Miracle on 34th Street, Speed, True Lies, Cliffhanger, Demolition Man, Free Willy, Jurassic Park, Aladdin, Batman, Returns, Death Becomes Her, Beauty and the Beast, Star Trek VI, Terminator II, Jetsons, The Abyss, StarQuest, Young Sherlock Holmes, Artificia Intelligence, Evolution, Minority Report, Mission: Impossible 2,Forces, ...

Games

- 1972 Pong
- 1978 Space Invaders
- 1980 Pacman
- 1985 Commodore Amiga
- 1988 Tetris
- 1989 Game boy

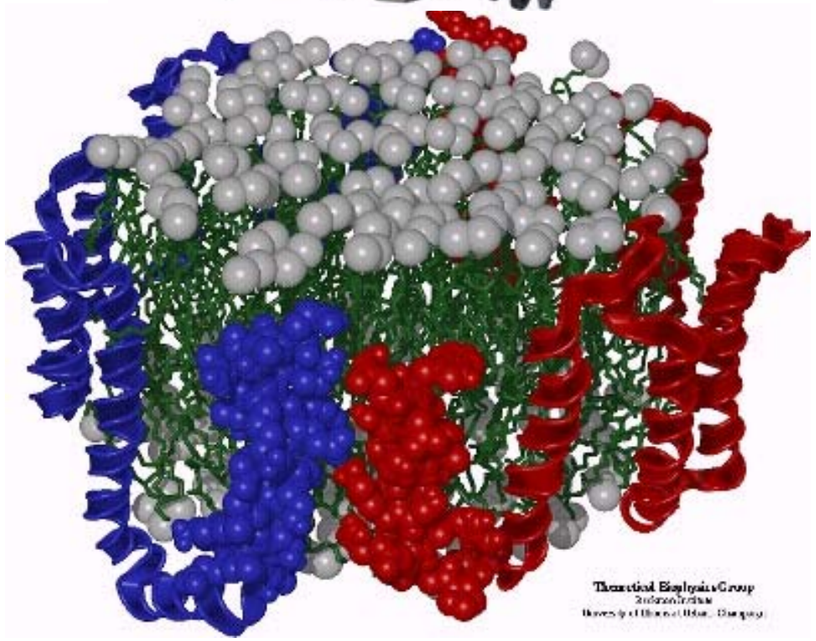




http://jamillan.com/primavera/crono.htm

Applications in different areas

- User interface
- CAD
- Medicine
- Scientific Visualization
- Simulation
- Leisure time
  - films, video games, ...



Contents of the course

Graphic libraries

- They are libraries with functions to render graphics
- Examples
  - OpenGL
  - Java3D
  - Phigs
  - DirectX
- Lenguages
  - WRML.

Transformations

- Coordinate systems
- 2D Transformations
- 3D Transformations
- Transformation composition
- Rotating about a pivot
- Rotating about an axis.

Model of 3D objects

- Objects building
- Rendering of objects
- Polygonal representation
  - structure, creation, mesh, attributes, ...
- Others methods (splines, CGS, volumetric).

Camera

- Coordinate systems and transformations
- Viewing coordinates
- Coordinate transformation matrix
- Projections
- Window and viewport.

Lighting

- Factors of light
- Reflection
- Ambient light
- Diffuse reflection
- Specular reflection
- Illumination models.

Color

- Light and color
- The visible light spectrum
- Primary and secondary colors
- Color spaces
  - RGB, CMY, YIQ, HLS, CIE
  - CIE XYZ, CIE xyY and CIE diagram properties.

Surface rendering methods

- Different surface rendering methods
- Wireframe
- Plane color
- Gouraud
- Phong
- Gouraud-Phong comparison.

Rendering algorithms

- Objective
- Rendering of lines
  - DDA algorithm
  - Bresenham algorithm
- Polygons rendering
- Polygons filling
- Visible surface detection.

Textures

- Concept of texture
- Use
  - Texture mapping
  - Environment mapping
  - Bump mapping.

Illumination models.

- Local illumination model and global illumination model
- The rendering equation of Kajiya
- Algorithms
  - Ray tracing
  - Radiosity
  - Radiance application.

Interaction

- The human senses
- Interaction systems
- Virtual reality
- Augmented reality.

Animation

- Capture and images sequences
- “Sprites” animation
- Key Frame animation
- 3D animation
- Examples:
  - Flash, Quicktime VR, VRML.

Sound

- Sound digitalization
- Sound process
- Compression
- Formats
- MIDI.

Multimedia graphics and Video

- Types of graphics, multimedia graphics, bitmaps, resolution, compression
- Video systems
- Digitalization of video, video standards, compression of video
- Streamed video
- Edition of digital video and process

Graphic Hardware

- Description of the elements
- Graphic cards
- Benchmarking
- Projection technologies.

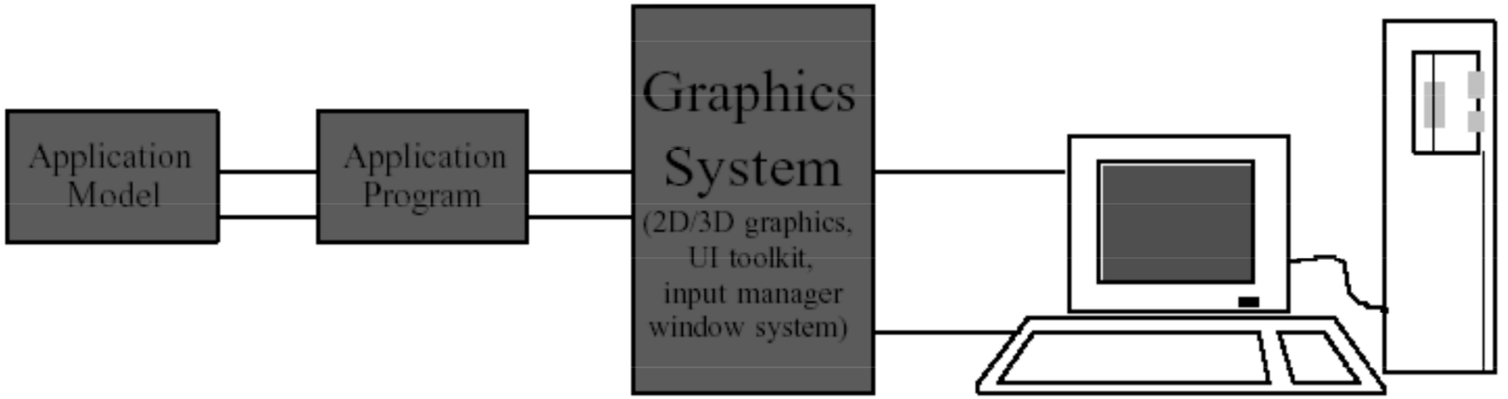
Graphic Libraries
Introduction to OpenGL

Introduction

- Graphic System:
  - Model + Visualization
- Visualization:
  - Use of specific hardware (3D)
  - Implemented through libraries

Definition of graphic library

- Software that generates images based on mathematical models and patterns of lighting, textures, etc.



Libraries, files, syntax

- Point of origin
  - IGL-Plot 10 (Tektronix)
  - Starbase (Hewlett Packard)
  - Iris GL Library (SGI)
- Distinguish from other graphic systems
  - VRML, X3D (Language of description)
  - DirectX-Direct3D
  - Java 3D
  - Open Inventor
  - Performer, Fahrenheit
  - Graphic engines

Objective of graphic libraries

- Independence of hardware (input devices as well as output devices).
- Independence of the application (the library is accessed through a unique interface (for each programming language) for any application.

Types of graphic libraries

- Direct Rendering and gfx packages:
  - OpenGL, Direct3D, GKS, PHIGS, PEX, GKS, etc…
- Scene-graph based
  - OpenGL Performer, Open Inventor, OpenGL Optimizer, PHIGS+, etc…
- Toolkits
  - World Toolkit, AVANGO, Game Engines, etc…

...

- Gestión imágenes 3D
  - “Bajo nivel”
    - Tareas
      - Gestión “en serie” de los elementos de la escena
      - Elementos de la escena
        - Primitivas gráficas
        - “Atributos” (≠ edition imágenes)
      - Variables de estado

        ↳ Generation imagen
    - Sistemas
      - OpenGL
      - Direct 3D
      - Java 3D
  - “Alto nivel” …

...

- (... Gestión imágenes 3D)
  - “Alto nivel”
    - Tareas
      - Gestión global de los elementos de la escena
      - Árbol escena
      - Carga/descarga de memoria
      - Gestión elementos no visibles
      - Election del modelo geométrico: Nivel detalle, Textura
      - Election de la técnica de presentation (rendering)
    - Sistemas
      - Inventor
      - Performer
      - (Fahrenheit)
      - Hewlett Packard

DirectX Componentes

- DirectDraw
- DirectSound
- DirectPlay
- Direct3D
- ….

DirectX - Introduction

- What is DirectX?
  - Set of APIs that allows developers of interactive content (image, video, sound ...) to have access to features of specialized hardware without writing a specific code for that hardware in MS Windows Systems.

DirectX - Introduction

- Components included in DirectX
  - It allows to develop applications of high performance and real time
    - **API** DirectPlay
    - **API** DirectInput
    - **API** DirectSound
    - **API** DirectDraw
    - **API** Direct3D

DirectX - Introduction

- Objectives of DirectX
  - Develop Windows applications of high performance using
    - Graphic cards
    - Plug´n Play
    - Communication services built in Windows
    - Resources of the system
  - Use of the new implemented hardware

DirectX - Introduction

- DirectX & COM (Component Object Model)
  - Object: black box that represents the hardware and requires communication with other applications through an interface.
  - Method: commands sent and received of the object through the COM interface
  - Ex.: Method **GetDisplayMode** is sent through the interface **IDirectDraw2** to get the current value of the resolution of the screen with the object **DirectDraw**

DirectX - DirectDraw

- It is in charge of managing the video memory
- Provide tools for:
  - Manipulation of multiple video buffers
  - Direct access to video memory
  - Page flipping
  - Back Buffering
  - Use of the graphic palette
  - Clipping

DirectX - DirectDraw

- Types of objects
  - IDirectDraw
  - IDirectDrawSurface
  - IDirectDrawPalette
  - IDirectDrawClipper
  - IDirectDrawVideoPort

DirectX - DirectDraw

- Graphic and Technical concepts:
  - Bitmaps
  - Surfaces of drawing (buffers)
  - Page Flipping and Back Buffering
  - Rectangles
  - Sprites
  - Video modes
  - Buffers
  - Overlays
  - Clippers
  - Video Ports

DirectX - DirectSound

- Components of Audio of DirectX:
  - Mix of audio channels
  - Hardware acceleration
  - Direct access to sound devices
  - Audio capture

DirectX - DirectSound

- COM Interfaces
  - IDirectSoundBuffer
  - IDirectSound3DBuffer
  - IDirectSound3DListener
  - IDirectSoundCapture
  - IDirectSoundCaptureBuffer

DirectX - DirectPlay

- Simplify the access of the applications to the communication services
- Grant independence for the creation of game servers
- Communications
  - Peer-to-Peer
  - Client/Server

DirectX - Direct3D

- Graphic interface for 3D hardware
  - It allows interactive tridimensional graphics in Windows applications.
- 2 Modes:
  - Immediate
    - Low level 3D API 3D
    - Independence of device
    - Experimented programmers
  - Retained
    - Fast developments
    - High layer of immediate mode

DirectX - Direct3D

- Basic concepts
  - 3D coordinate systems
    - Left-handed
    - We can simulate right-handed
  - Transformations 3-D
    - Translation
    - Rotation
    - Scale up / scale down
  - Polygons
    - Normal to the face and vertex
    - Shading modes
    - Triangles interpolation
  - Triangles
  - Rules to render triangles

DirectX - Direct3D

- Conceptos Básicos
  - Triangles
  - Rules to render triangles

OpenGL

- Introduced in 1992 by SGI
- Based in IRIS GL, an API for SGI workstations
- It is an *open standard* that has been widely adopted for all kind of graphic applications
- It is developed under the supervision of the **OpenGL architecture review board**

Design objectives of OpenGL:

- Graphic API of high performance (with hardware acceleration)
- It has some independence of the hardware
- It is a natural API in C with possibilities of extensibility

It has become a standard because …

- It doesn’t try to do many things:
  - Only render the image, doesn’t manage windows, etc...
  - It doesn’t have high level animation, model, sound, etc...
- It does what is needed:
  - Useful render effects and high performance
- It was promoted by leader companies such as SGI, Microsoft, etc

Advantages of OpenGL (1)

- **Industry standard**
  An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.
- **Stable**
  OpenGL implementations have been available for more than **seven years** on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.
- **Reliable and portable**
  All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

Advantages of OpenGL (2)

- **Evolving**
  Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.
- **Scalable**
  OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

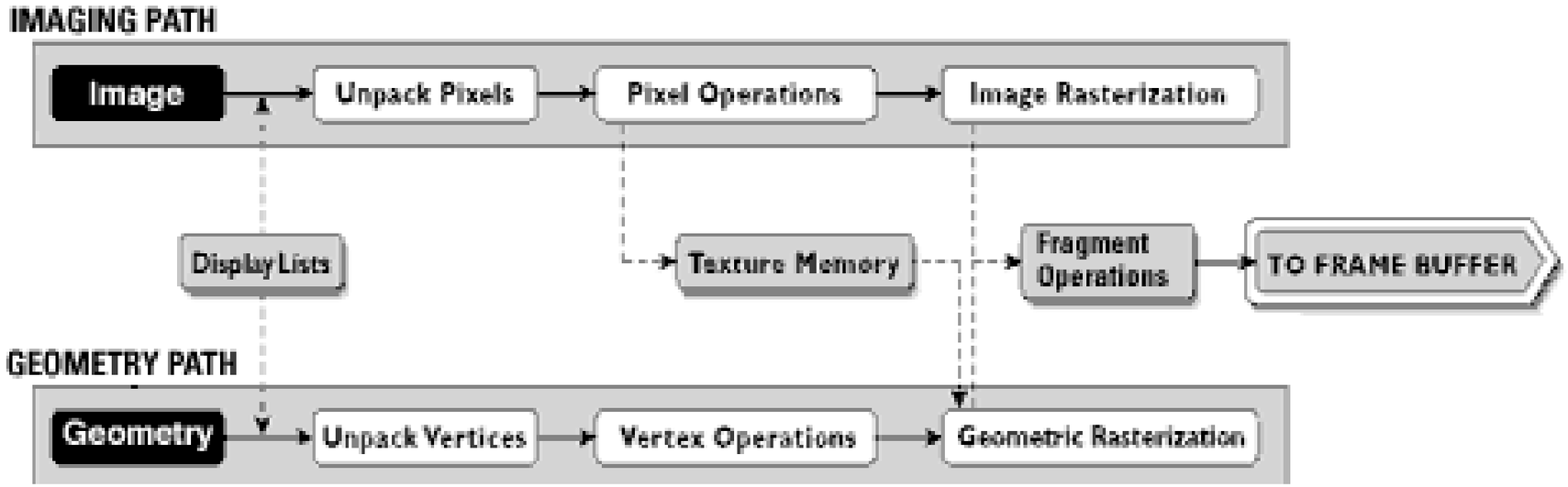
Advantages of OpenGL (3)

- **Easy to use**
  OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.
- **Well-documented**
  Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

Render of OpenGL

- Geometric primitives:
  - Points, lines and polygons
- Images primitives:
  - Images and bitmaps
- Differentiated pipelines for images and geometry, joined by the texture mapping
- The render depends on the state (lights, colors, materials, etc)

OpenGL Architecture



OpenGL features (1)

- **Accumulation buffer** A buffer in which multiple rendered frames can be composited to produce a single blended image. Used for effects such as depth of field, motion blur, and full-scene anti-aliasing.
- **Alpha blending**. Provides a means to create transparent objects.
- **Automatic rescaling of vertex normals** changed by the modeling matrix.
- **BGRA pixel formats and packed pixel formats** to directly support more external file and hardware frame buffer types.
- **Color-index mode**. Color buffers store color indices rather than red, green, blue, and alpha color components.
- **Immediate mode**. Execution of OpenGL commands when they're called, rather than from a display list.
- **Display list**. A named list of OpenGL commands. The contents of a display list may be preprocessed and might therefore execute more efficiently than the same set of OpenGL commands executed in immediate mode.

OpenGL features (2)

- **Double buffering**. Used to provide smooth animation of objects. Each successive scene of an object in motion can be constructed in the back or "hidden" buffer and then displayed. This allows only complete images to ever be displayed on the screen.
- **Feedback**. A mode where OpenGL will return the processed geometric information (colors, pixel positions, and so on) to the application as compared to rendering them into the frame buffer.
- **Level of detail control** for mipmap textures to allow loading only a subset of levels.
- **Materials lighting and shading**. The ability to accurately compute the color of any point given the material properties for the surface.
- **Pixel operations**. Storing, transforming, mapping, zooming.
- **Polynomial evaluators**. To support non-uniform rational B-splines (NURBS).
- **Primitives**. A point, line, polygon, bitmap, or image.
- **Raster primitives**. Bitmaps and pixel rectangles.

OpenGL features (3)

- **RGBA mode**. Color buffers store red, green, blue, and alpha color components, rather than indices.
- **Selection and picking**. A mode in which OpenGL determines whether certain user-identified graphics primitives are rendered into a region of interest in the frame buffer.
- **Specular Highlights**. Application of specular highlights after texturing for more realistic lighting effects.
- **Stencil planes**. A buffer used to mask individual pixels in the color frame buffer.
- **Texture coordinate edge clamping** to avoid blending border and image texels during texturing.

OpenGL features (4)

- **Texture mapping**. The process of applying an image to a graphics primitive. This technique is used to generate realism in images.
- **Three Dimensional Texturing**. Three-dimensional texturing for supporting hardware-accelerated volume rendering.
- **Transformation**. The ability to change the rotation, size, and perspective of an object in 3D coordinate space.
- **Vertex array enhancements** to specify a subrange of the array and draw geometry from that subrange in one operation.
- **Z-buffering**. The Z-buffer is used to keep track of whether one part of an object is closer to the viewer than another.

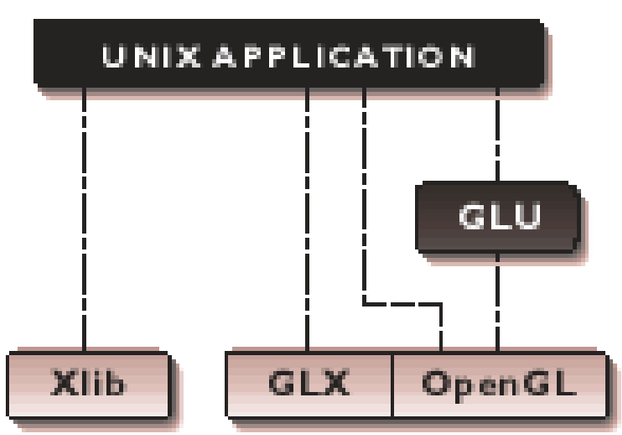
Related APIs

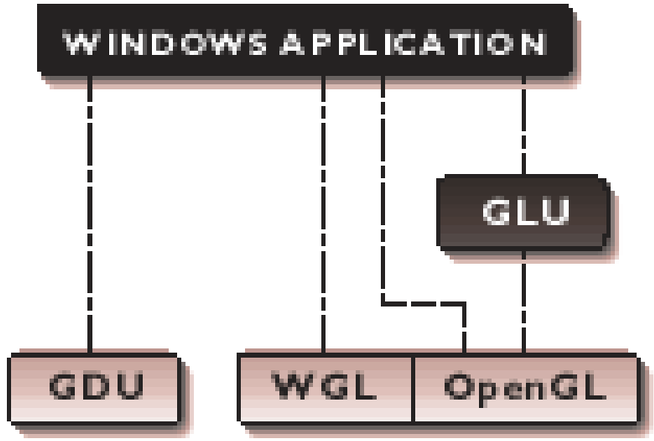
- GLX, WGL, AGX
  - Connection of OpenGL with the window environment
- GLU (OpenGL Utility library)
  - It is part of the OpenGL
  - It includes support for quadratic, NURBS, etc.
- GLUT (OpenGL Utility Toolkit)
  - It is not a official part of the OpenGL
  - It allows the portability of the applications on different window systems
- MESA: an OpenGL clone

OpenGL APIs

- Library of functions to generate images from 3D models, plus other auxiliary libraries
  - gl: the OpenGL library that interacts with the hardware
  - glu: library of higher level, built upon OpenGL
  - glaux: not used currently
  - glut library to build transportable user interfaces (Linux, Windows, Unix, MacOS)
    - glut32.dll → windows\system(32)
    - glut32.lib → DevStudio\Vc\lib
    - glut.h → DevStudio\Vc\include\gl

Architecture of the OpenGL APIs





Programming details

- Add libraries opengl32.lib glu32.lib glut32.lib
  - Project : settings : link : Object/Library Modules
- Files to include
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - If you are using GLUT for managing your window manager
    - #include <GL/glut.h>
    - Note that glut.h includes gl.h, glu.h, and glx.h automatically, so including all three files is redundant.

States

- Machine of states
  - Background color
  - Light intensity
  - Material
  - Switch on – switch off light
  - etc
- Value or state
  - glColor*(), glGetFloatv()
  - glEnable(), glDisable(), glIsEnabled()

Command syntax (functions)

- glVertex3fv(...)
  - **gl** tells that this function belongs to the “gl” s/w package
  - **3** is used to indicate three arguments
  - **f** is used to indicate that the arguments are floating point
  - **v** indicates that the arguments are in vector format
- Number Of Arguments: 2, 3, or 4
  - Bi-dimensional version of the command
  - 3D or rgb
  - Homogeneous coordinates or rgb+alpha
- Formats
  - absence of v indicates scalar format
  - v indicates vector format

Variable types and constants

- The most equivalences appear in the table
  - It is recommended to define the arguments that are passed to the OpenGL functions using their predefined types.
    - For Glint some systems may use “short”, others “long”
    - For Glfloat some systems may use “float” others “double”

	Data type	**Typical Corresponding C-Language Type**	**OpenGL Type Definition**
b	8-bit integer	C-Language Type	GLbyte
s	16-bit integer	OpenGL Type	GLshort
i	32-bit integer	int or long	Glint GLsizei
f	32-bit floating-point	float	Glfloat GLclampf
d	64-bit floating-point	double	Gldouble GLclampd
ub	8-bit unsigned integer	unsigned char	Glubyte GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	Gluint Glenum GLbitfield

Constants

- Example : GL_COLOR_BUFFER_BIT
- Defined constants
  - Begin with GL_
  - Use all capital letters
  - Use underscores to separate words
- Very frequently operations with “or” are used

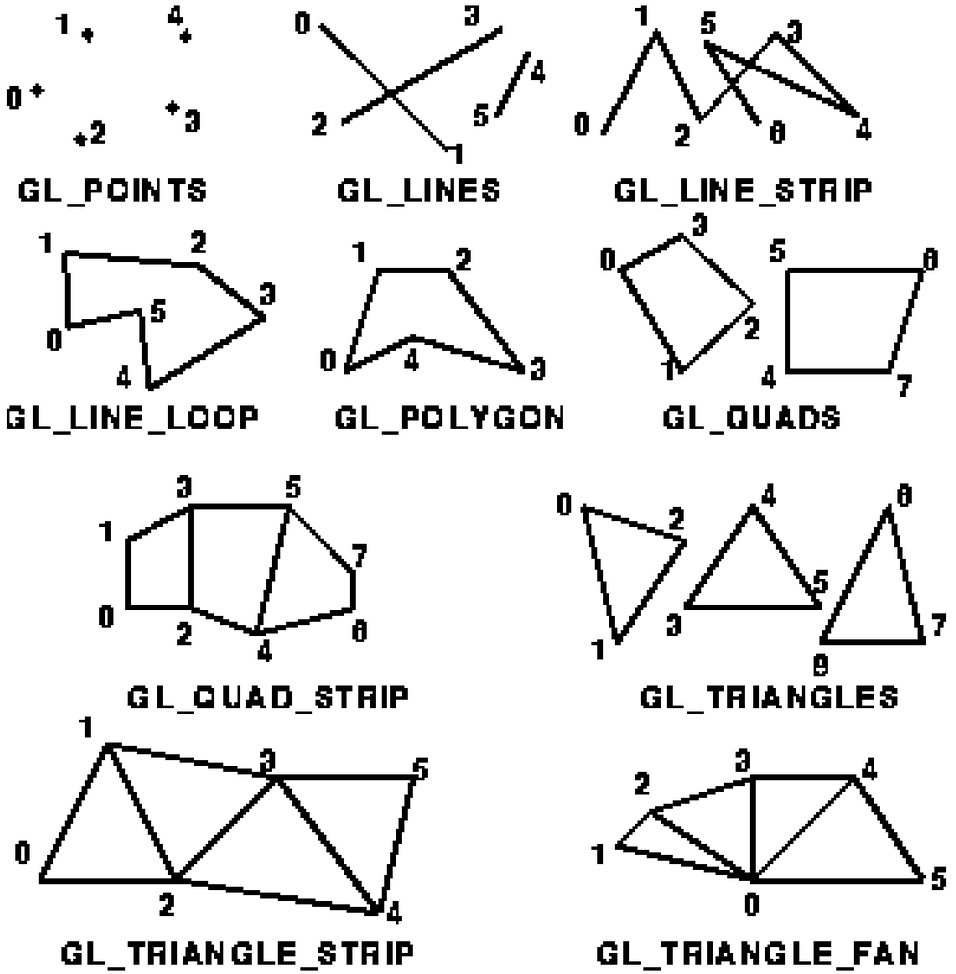
... (gl)

- pure "output", but lacks connection with display
  - glClear (command)
  - glClearColor (state)
  - glBegin glEnd
  - glVertex* (geometry)
  - glColor* (attribute, state)
  - glFlush, glFinish (command stack & processing)

… (glut)

- Window system independent management
  - “window” management
    - glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    - glutInitWindowSize(WIDTH, HEIGHT);
    - glutInitWindowPosition (550, 350);
    - glutCreateWindow(“Basic Draw");
  - “input” : event handling
    - similar to : The X Window system, MOTIF, MFF, ...
    - (event)
    - void Display(void); (callback function)
    - glutDisplayFunc(*Display*); (binding event & callback)
      - **void** *Display* (**void**)
    - glutMainLoop(); (dispatching events)

Primitives

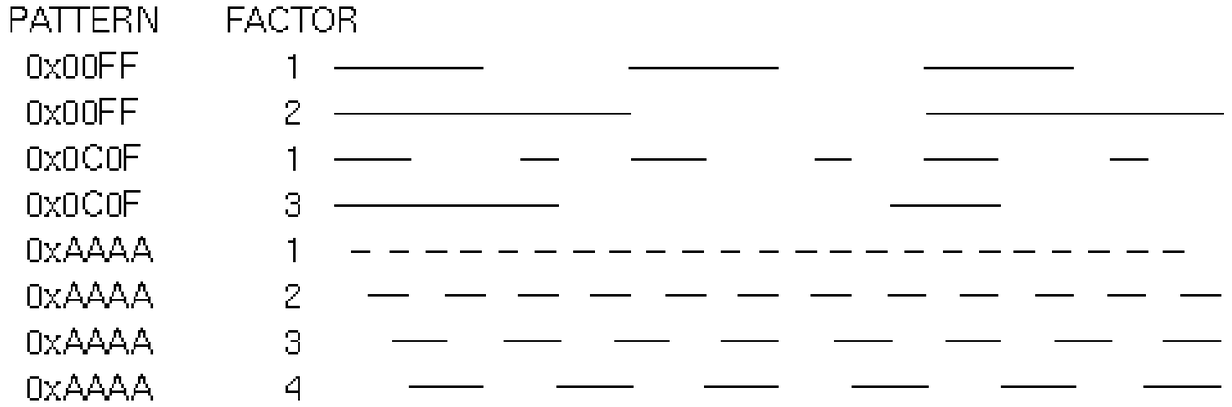

GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_POLYGON
GL_QUADS
GL_QUAD_STRIP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN

...

- Other available primitives
  - Objects (auxiliary purpose, not for build models)
  - Vertex arrays
  - Display lists
  - Evaluators, NURBS, etc
- Text
  - There isn’t a primitive
  - It is needed to use textures, with aliasing problem

Attributes

- glPointSize(GLfloat)
- glLineWidth(GLfloat)
- glLineStipple(GLint factor, GLushort pattern)
  - glEnable(GL_LINE_STIPPLE)
- glPolygonMode(face, mode)
  - GL_FRONT GL_BACK *GL_FRONT_AND_BACK*
  - GL_POINT GL_LINE *GL_FILL*
- glPolygonStipple (enable) “**transparency**”
- glEdgeFlag* **mode Line, splitted concave polygons**
- glColor*
- glMaterial*



State variables

- Defect values
- Current value
- Management in an application or in a reusable module
  - Defect value in an application
  - Policy of state change in a software module
- glPushAttrib(ored mask) / glPopAttrib()
  - It allows to save a group of attributes

Groups of attributes

GL_ACCUM_BUFFER_BIT	accum-buffer
GL_ALL_ATTRIB_BITS	--
GL_COLOR_BUFFER_BIT	color-buffer
GL_CURRENT_BIT	current
GL_DEPTH_BUFFER_BIT	depth-buffer
GL_ENABLE_BIT	enable
GL_EVAL_BIT	eval
GL_FOG_BIT	fog
GL_HINT_BIT	hint
GL_LIGHTING_BIT	lighting
GL_LINE_BIT	line
GL_LIST_BIT	list
GL_PIXEL_MODE_BIT	pixel
GL_POINT_BIT	point
GL_POLYGON_BIT	polygon
GL_POLYGON_STIPPLE_BIT	polygon-stipple
GL_SCISSOR_BIT	scissor
GL_STENCIL_BUFFER_BIT	stencil-buffer
GL_TEXTURE_BIT	texture
GL_TRANSFORM_BIT	transform
GL_VIEWPORT_BIT	viewport

Grouping

- Arrays
  - It avoids wasting time in calling functions
- Display Lists
  - Idem
  - The information can be preprocessed
  - It is stored in the graphic processor to relieve the communication bus
    - Workstations
    - PCs

Commands between: glBegin-glEnd

Command	Purpose of Command	Reference
glVertex*()	set vertex coordinates	Chapter 2
glColor*()	set current color	Chapter 5
glIndex*()	set current color index	Chapter 5
glNormal*()	set normal vector coordinates	Chapter 2
glEvalCoord*()	generate coordinates	Chapter 11
glCallList(), glCallLists()	execute display list(s)	Chapter 4
glTexCoord*()	set texture coordinates	Chapter 9
glEdgeFlag*()	control drawing of edges	Chapter 2
glMaterial*()	set material properties	Chapter 6

Example : Window to viewport

- gl
  - glMatrixMode
  - glLoadIdentity
  - viewport
  - glOrtho
  - glGetIntegerv
- glut
  - glutReshapeFunc(myNewSize)
    - void myNewSize(int w, int h)

Interesting links

- www.opengl.org Official site of OpenGL
- nehe.gamedev.net NeHe, various OpenGL tutorials
- romka.demonews.com Romka, various OpenGL tutorials
- Nexe.gamedev.net same as Nehe for DirextX

Transformations

Content

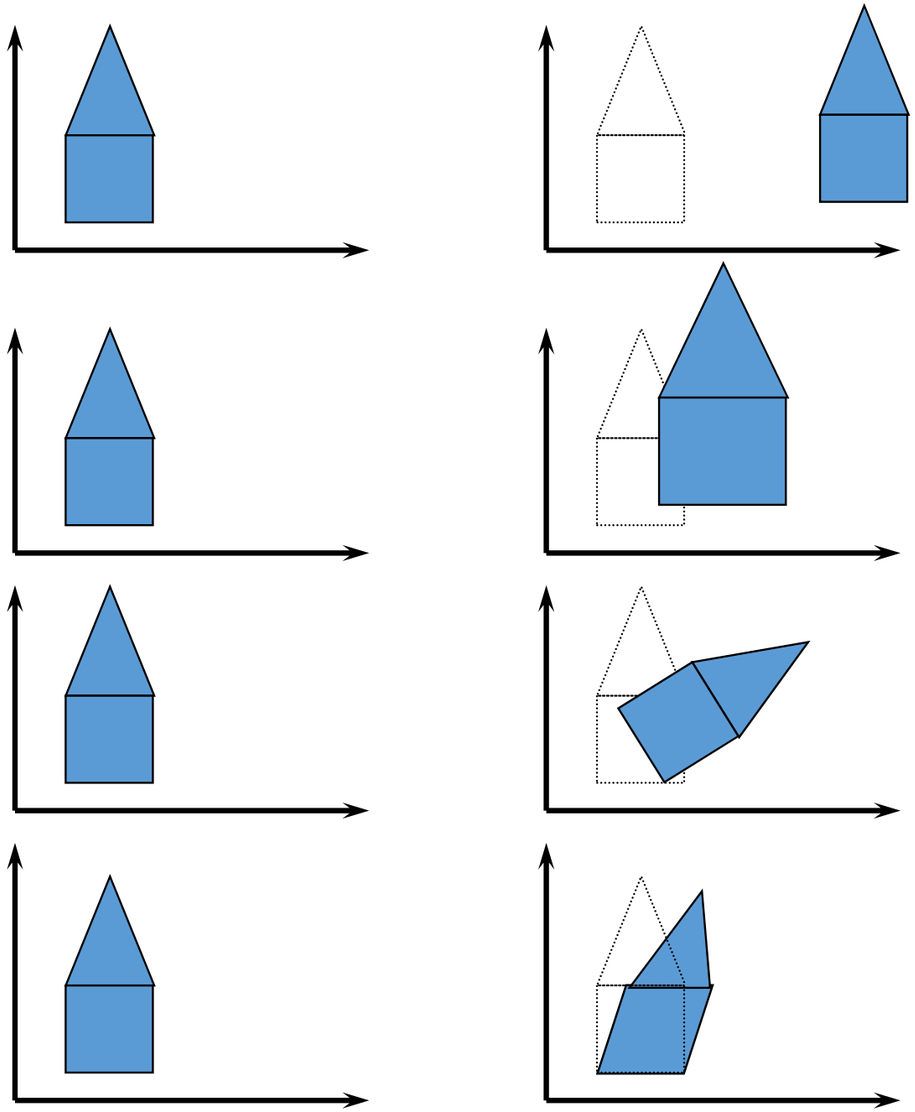
- Coordinate systems
- 2D Transformations
- 3D Transformations
- Transformation composition
- Rotating about a pivot
- Rotating about an axis

Agradecimientos:
A Alex García-Alonso por facilitar el material para la realization de estas transparencias (http://www.sc.ehu.es/ccwgamoa/clases)

Coordinate systems

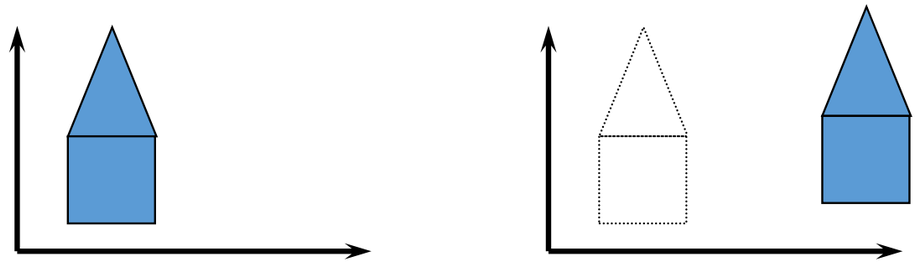
- An object is represented by polygons
- A polygon is a collection of vertex and edges
- To transform an object we must transform their vertex
- From local system to the global system: transformations

2D Transformations



- Translation
- Scale
- Rotation
- Deformation

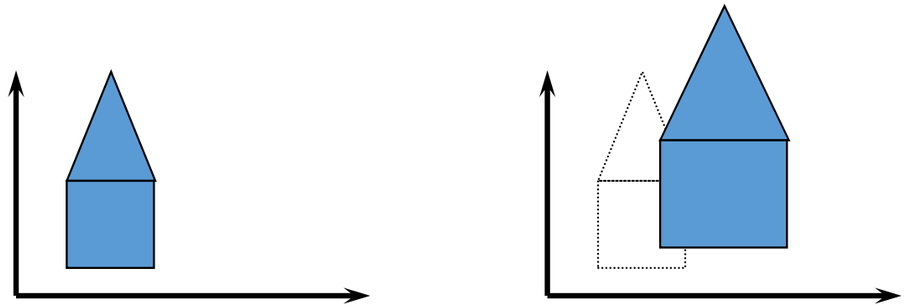
2 dimensions: translation



$$x' = x + t_x$$
$$y' = y + t_y$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

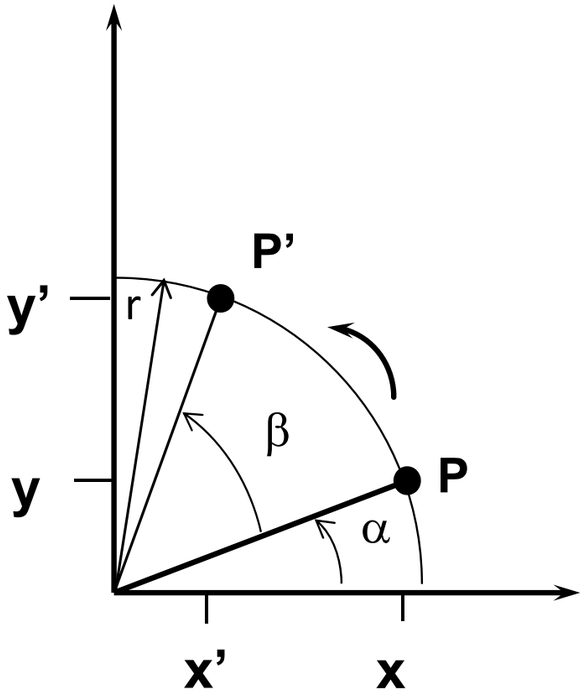
2 dimensions: scale



$$x' = s_x \cdot x$$
$$y' = s_y \cdot y$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2 dimensions: rotation



$x = r \cos \alpha$

$y = r \operatorname{sen} \alpha$

$x' = r \cos (\alpha + \beta) =$

$= r (\cos \alpha \cos \beta - \operatorname{sen} \alpha \operatorname{sin} \beta) =$

$= x \cos \beta - y \operatorname{sen} \beta$

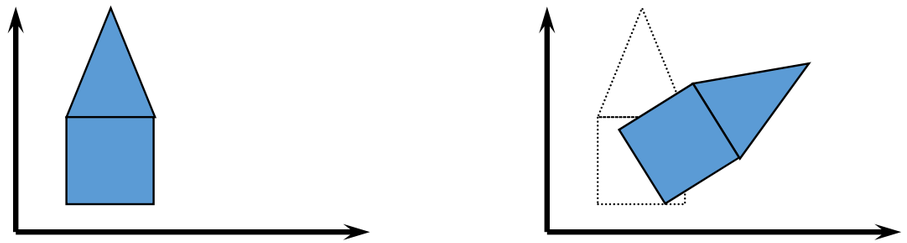
$y' = r \operatorname{sen} (\alpha + \beta) =$

$= r (\cos \alpha \ \operatorname{sen} \beta + \operatorname{sen} \alpha \cos \beta) =$

$= x \operatorname{sin} \beta + y \cos \beta$

2 dimensions: rotation

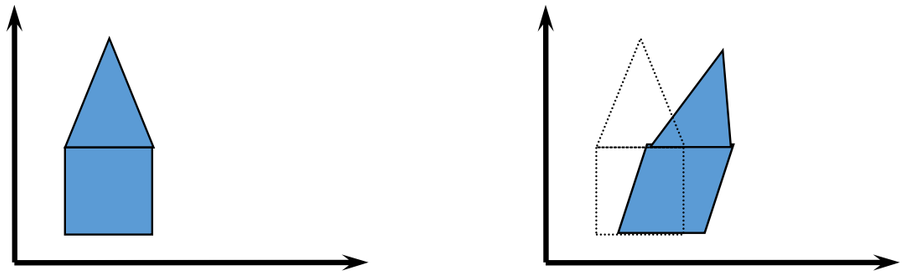
- Matricially representation with Homogenous coordinates:



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2 dimensions: deformation (shear)

- Deformation of x coordinate:



$$x' = x + h_x \cdot y$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & h_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3D Transformations

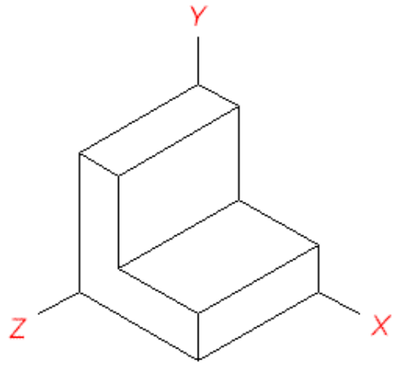
- The general expression of a 3D transformation in Homogenous coordinates:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

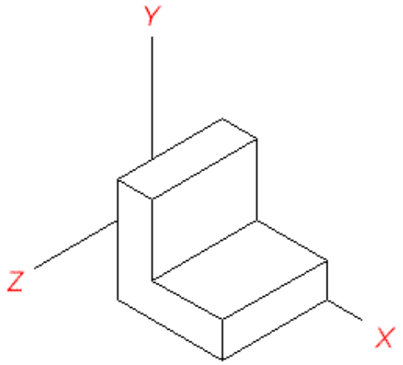
Transformation Matrix M_{44}

- Describe all the transformations: translation, scale, rotation, deformation.
- The composition of transformations is made by the product of matrix
- You can get the values of the transformation from the matrix: displacement, scale y turn.

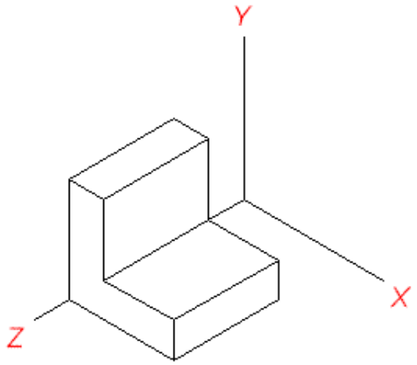
3D: Translation



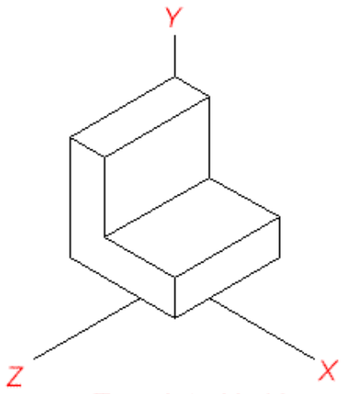
Original object position



Translated in X



Translated in Z

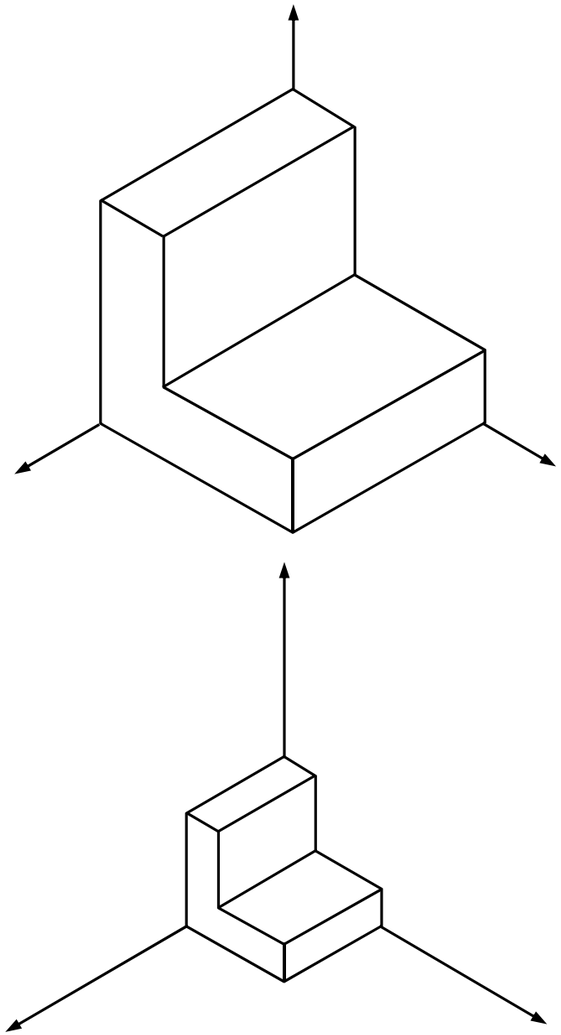


Translated in Y

$$
\begin{aligned}
x' &= x + t_x \\
y' &= y + t_y \\
z' &= z + t_z
\end{aligned}
$$

$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

3D: Scale

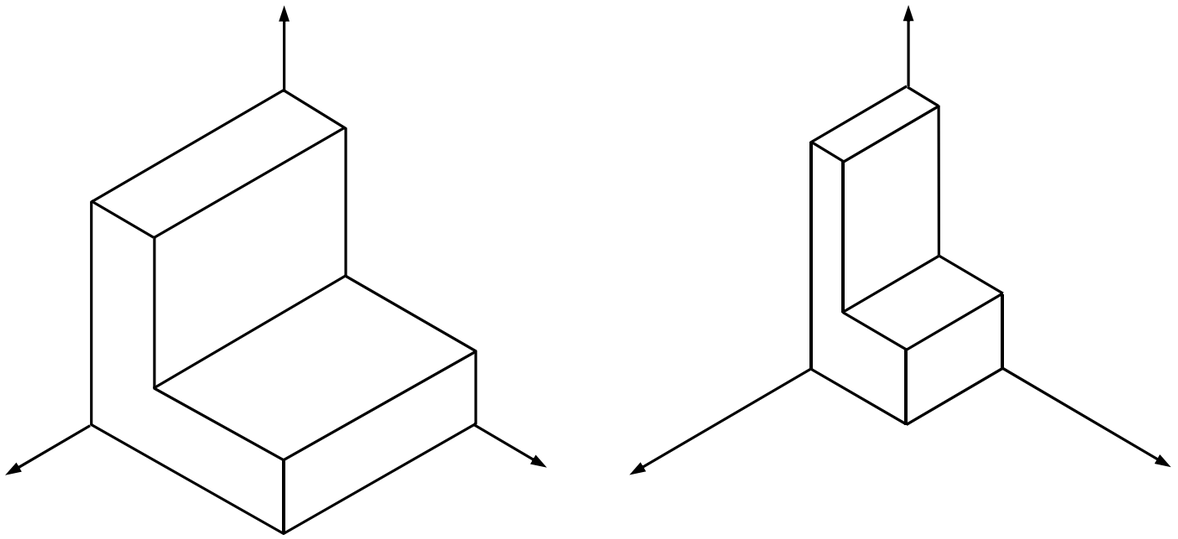


$$
\begin{aligned}
x' &= s_x \cdot x \\
y' &= s_y \cdot y \\
z' &= s_z \cdot z
\end{aligned}
$$

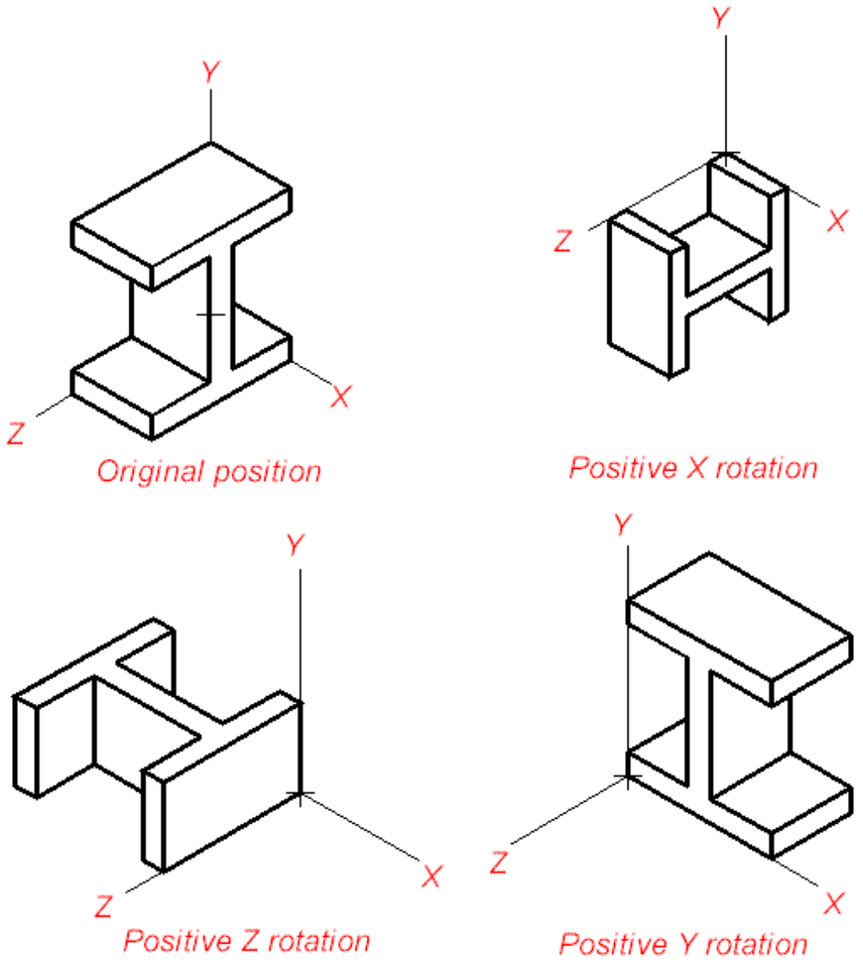
$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

3D: Non uniform scale

$s_x \neq s_y \neq s_z$



3D: Rotation


Y
Z
X
Original position
Y
Z
X
Positive X rotation
Y
Z
X
Positive Z rotation
Y
Z
X
Positive Y rotation

3D: Rotation matrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

X Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Y Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Z Rotation

Others transformations

Shear in xy (z invariant)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

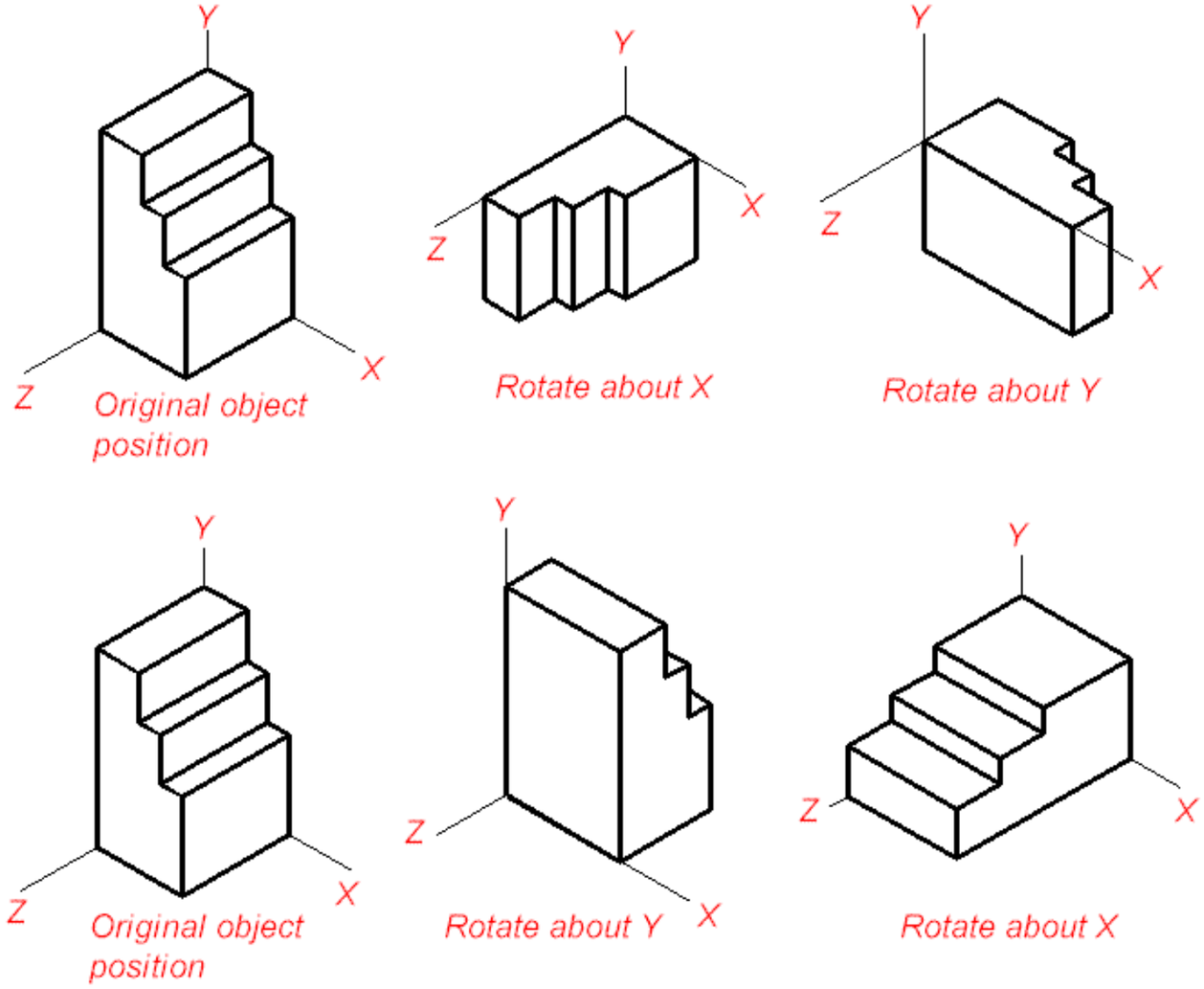
Reflexion on plane xy

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Composite transformation

- The transformations can be applied sequentially to a point.
  - The result of the first transformation:
    - $M_1 \cdot P$
  - The second transformation:
    - $M_2 \cdot [M_1 \cdot P] = [M_2 \cdot M_1] \cdot P$
- The composite transformation is made by the product of the matrix
    - $M = M_n \cdot M_{n-1} \cdot \ldots \cdot M_2 \cdot M_1$

Transformation product may not be commutative



Hierarchic structure

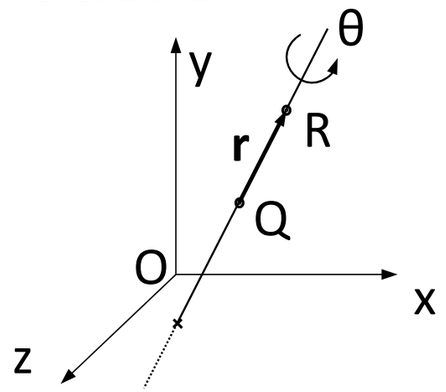
- An object is positioned in its coordinate system.
- All the assembly can be positioned in another coordinate system and so successively.
- The coordinates in the final system are got by the composite transformation.

Pivot-point rotation

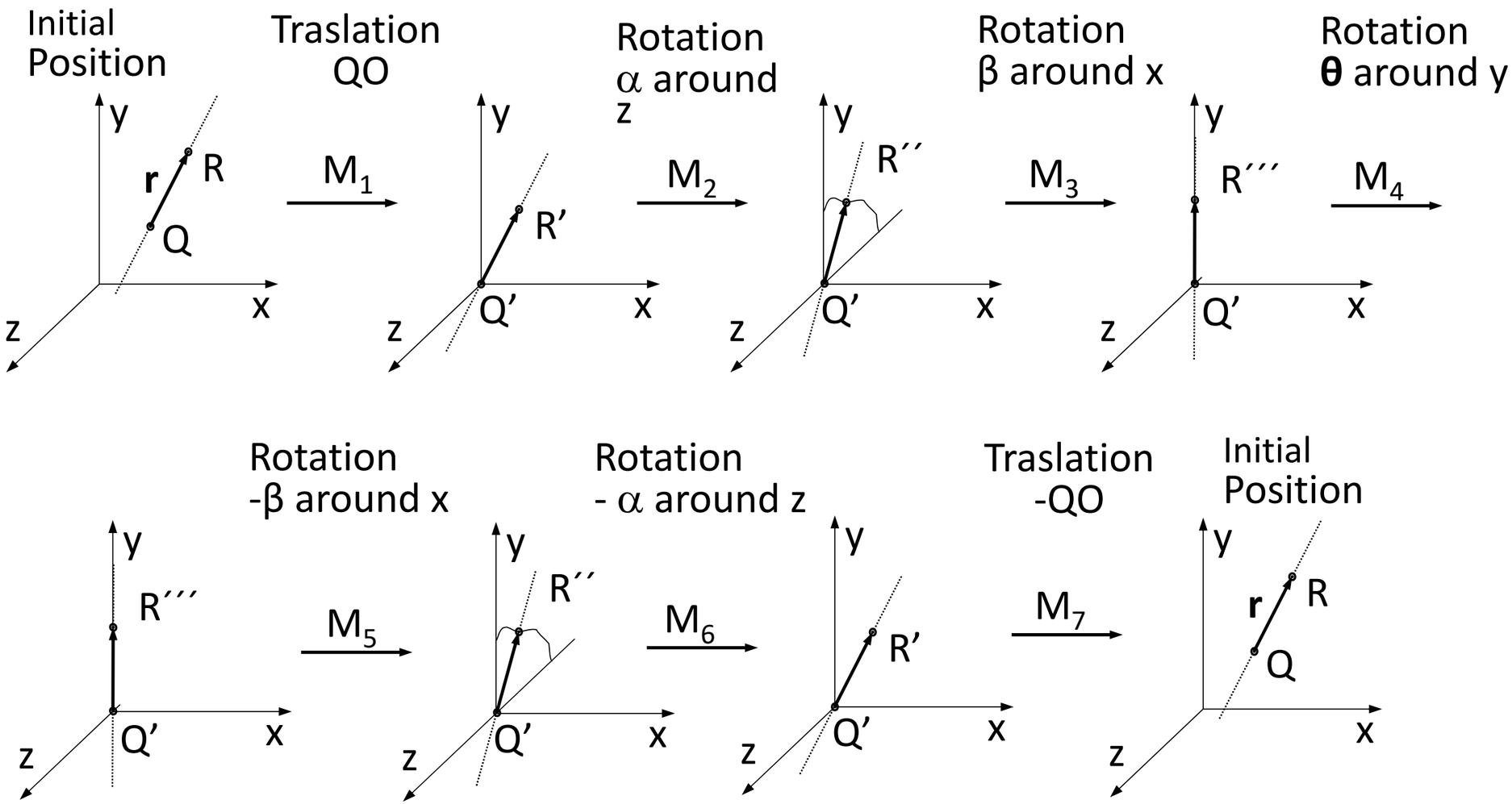
- When the axis doesn't cross the coordinate origin, the next operations are required:
  - Translate the pivot-point Q, to the coordinate origin
  - Rotate the object
  - Make the inverse translation
- The composite transformation matrix is:
  - $M_{R_Q}(\theta) = M_3 \cdot M_2 \cdot M_1$
  - $M_{R_Q}(\theta) = M_T(q_x, q_y, q_z) \cdot M_R(\theta) \cdot M_T(-q_x, -q_y, -q_z)$
- The fixed point scaling is done in the same way

Rotation around an axis

- The axis is defined by a point “Q” and a unit vector “r”. The rotation has a value defined by angle θ.
- It is solved with a composition of transformations
  - Definition of transformations
  - Calculation of each one
  - Explanation of calculation of required angles



Rotation around an axis : composition of transformations

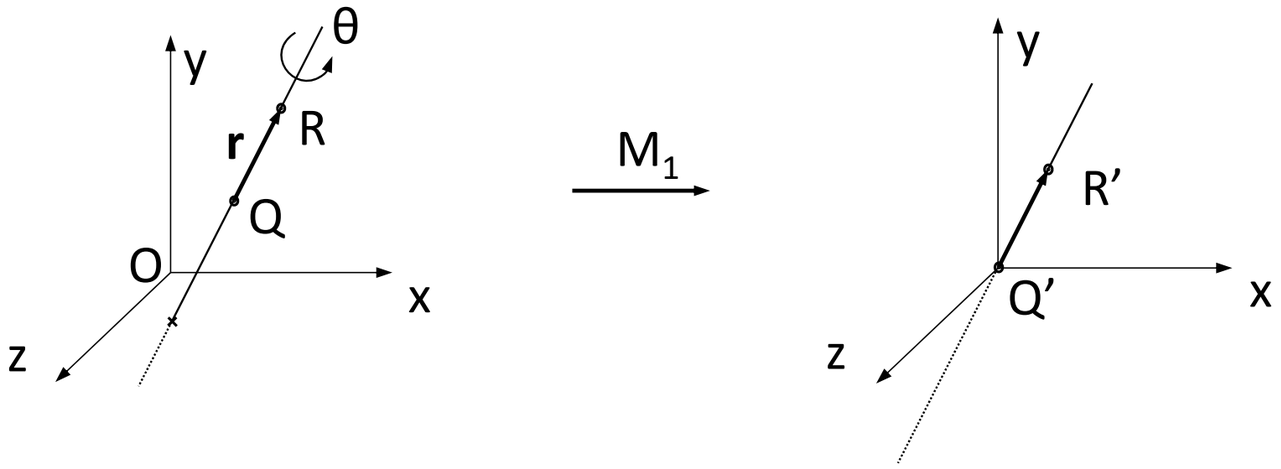


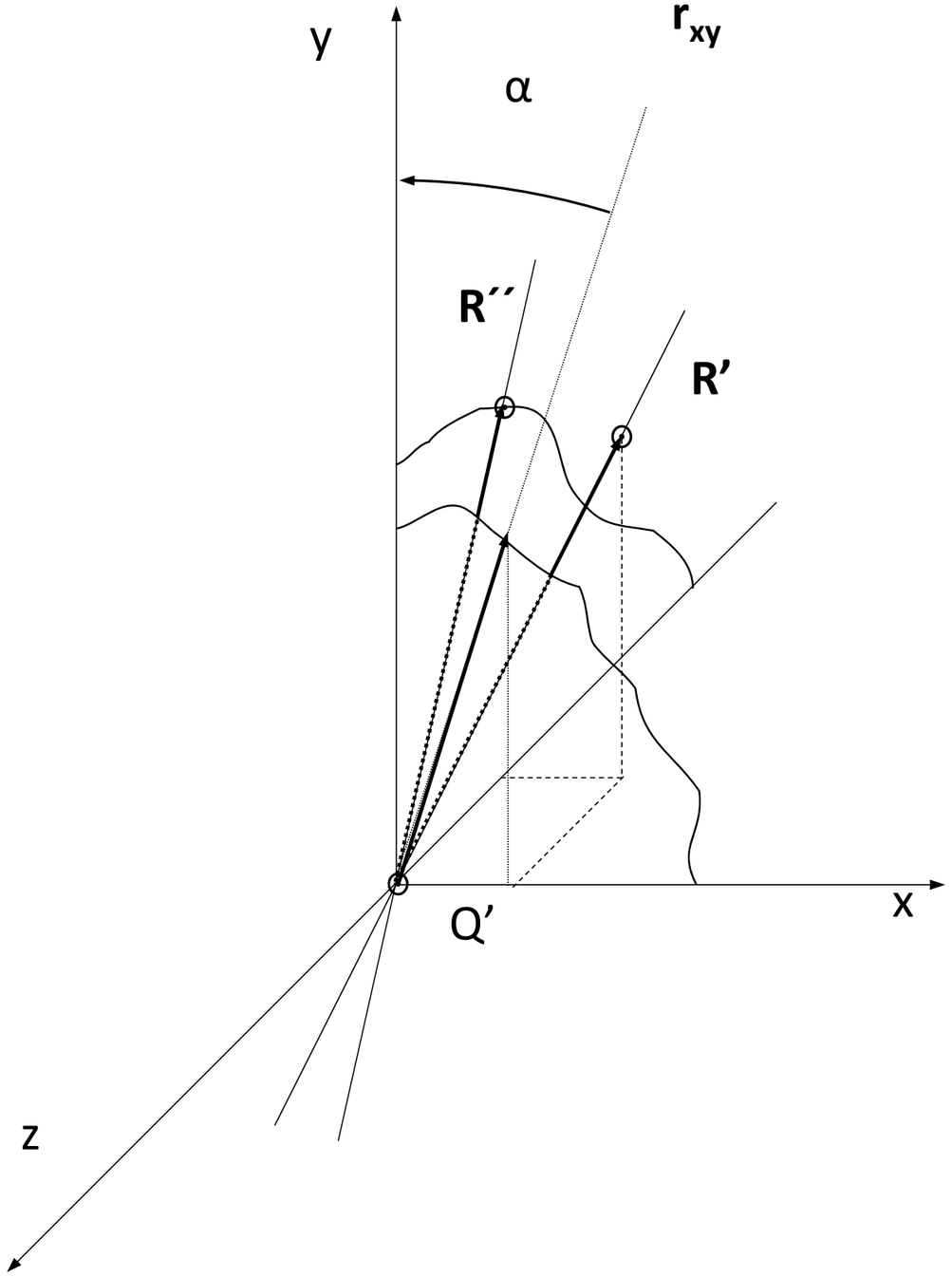
Rotation around an axis: relation of transformations

- The transformation matrix is:
  - $M_{(Q,\mathbf{r})}(\theta) = M_7 \cdot M_6 \cdot M_5 \cdot M_4 \cdot M_3 \cdot M_2 \cdot M_1$
  - M_1 : traslation QO
  - M_2 : rotation α around z
  - M_3 : rotation β around x
  - M_4 : rotation θ around y
  - M_5 : rotation $-\beta$ around x
  - M_6 : rotation $-\alpha$ around z
  - M_7 : traslation -QO

Rotation around an axis : M_1 - traslation QO

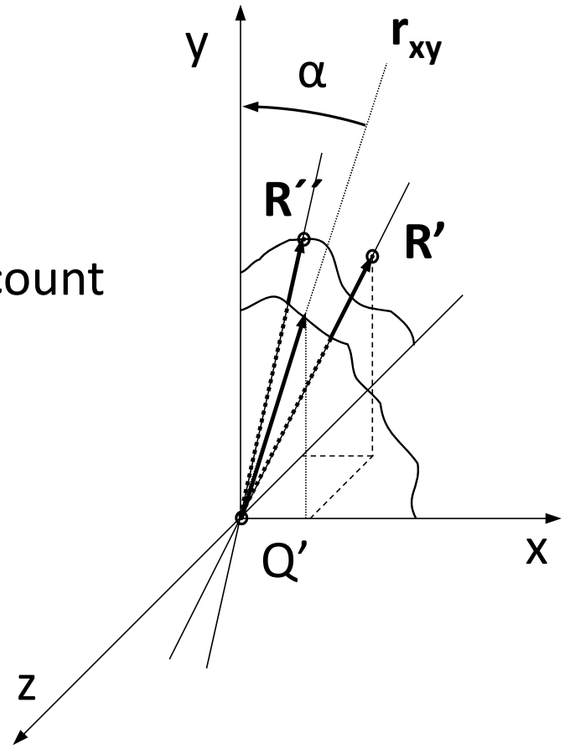
- We define R so OR = OQ + r
- The translation that moves Q to the origin is:
  - $M_1 = M_T(-q_x, -q_y, -q_z)$




y
r_{xy}
α
R´´
R’
Q’
x
z

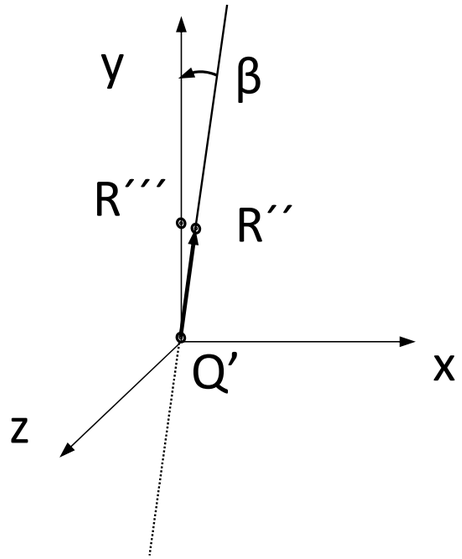
Rotation around an axis: M_2 - rotation α around z

- Calculation of the α angle between the planes YZ and the plane defined by the z axis and OR'
  - $\mathbf{r_{xy}}$ is the orthogonal projection of r on XY
  - R'' is the result of rotating around z
  - α is the angle between r_{xy} and j (unit vector of y)
  - The positive direction of k must be taken into account
- M_2 is the rotation matrix around the z axis:
  - $M_2 = M_{Rz}(\alpha)$



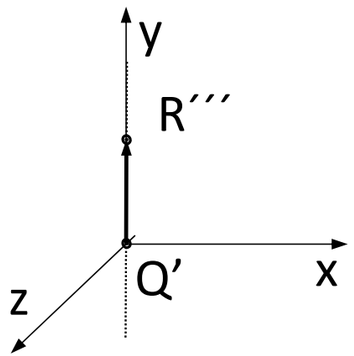
Rotation around an axis: M_3 - rotation β around x

- Applying M_2 to R’ we get R’’
  - R’’ is on the YZ plane
  - r’’ is defined by OR’’
- Calculation of the β angle between r’’ and j
  - The positive direction of i must be taken into account
- M_3 is the rotation matrix around the x axis:
  - $M_3 = M_{Rx}(\beta)$



Rotation around an axis: M_4 - rotation θ around y

- Applying M_3 to R’’ we get R’’’
  - R’’’ is on the y axis
- The rotation θ is done around the y axis
- M_4 is the rotation matrix around the y axis:
  - $M_4 = M_{Ry}(\theta)$

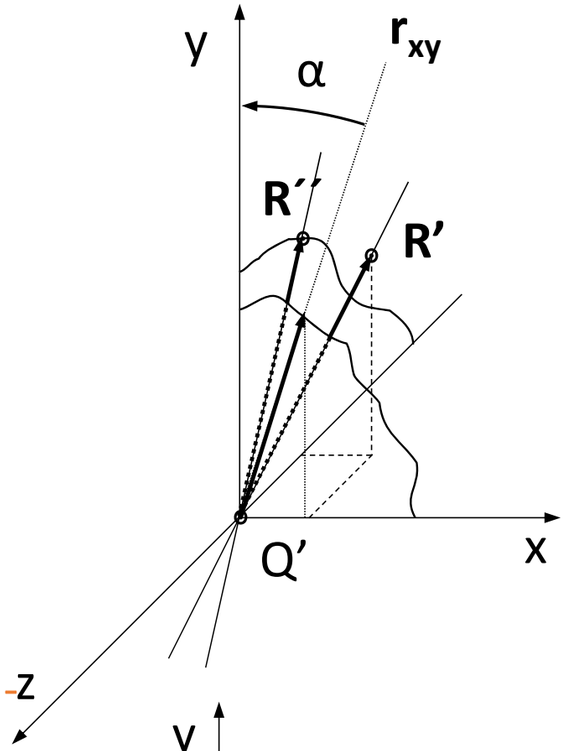


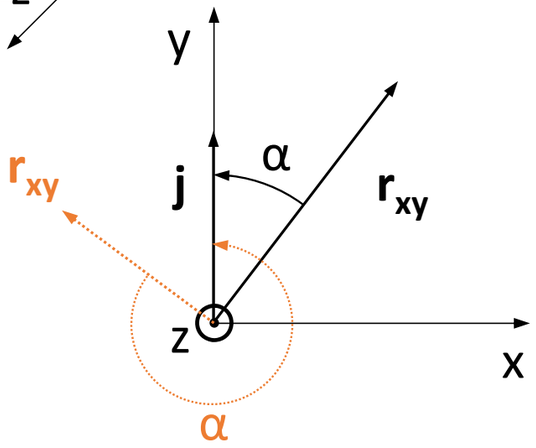
Rotation around an axis: M_5, M_6, M_7 - inverses

- After calculating the θ rotation around the transformed axis, it is necessary to invert the transformation process. To do it, the inverses matrices have to be calculated.
  - $M_5 = M_{Rx}(-\beta)$
  - $M_6 = M_{Rz}(-\alpha)$
  - $M_7 = M_T(q_x, q_y, q_z)$
- The composed transformation matrix is:
- $M_{(Q,\mathbf{r})}(\theta) = M_7 \cdot M_6 \cdot M_5 \cdot M_4 \cdot M_3 \cdot M_2 \cdot M_1$

Rotation around an axis: α angle

- Calculation of α angle
  - $\mathbf{r_{xy}}$ is the orthogonal projection of r on the XY plane: $(r_x, r_y, 0)$
  - $\cos \alpha = j \cdot \mathbf{r_{xy}} / | \mathbf{r_{xy}} | =$
    $= ((0, 1, 0) \cdot (r_x, r_y, 0)) / (r_x^2 + r_y^2)^{1/2}$
  - $\cos \alpha = r_y / (r_x^2 + r_y^2)^{1/2}$
  - As $\cos \alpha = \cos (-\alpha)$, then
    - if $r_x < 0$ then the angle must be $(2\pi - \alpha) \Rightarrow \alpha = -\alpha$

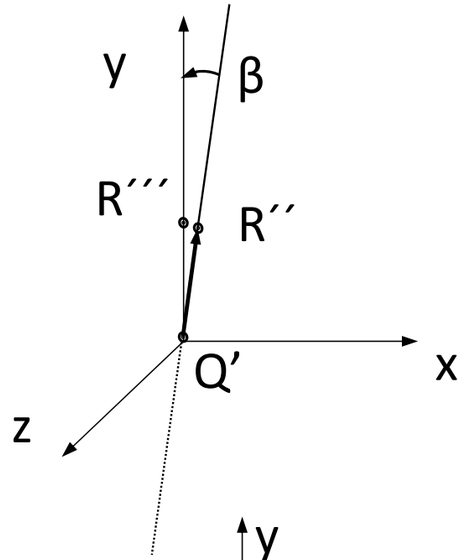


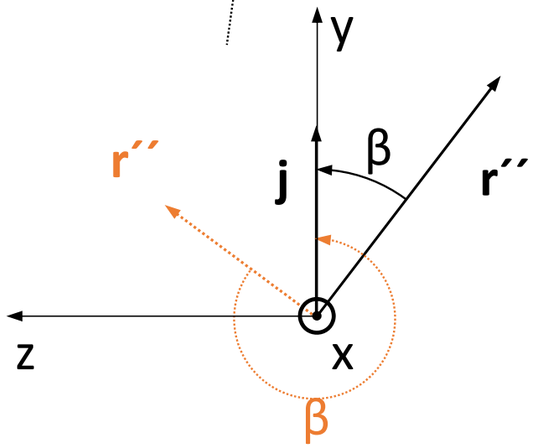


$$\alpha = \text{acos} (r_y / (r_x^2 + r_y^2)^{1/2})$$
$$\text{y si } r_x < 0 \Rightarrow \alpha = -\alpha$$

Rotation around an axis: β angle

- Calculation of β angle
  - R'' and r'' are on the YZ plane
  - $\cos \beta = j \cdot r'' / | r'' | =$
    $= ((0, 1, 0) \cdot (0, r_y'', r_z'')) / (r_y''^2 + r_z''^2)^{1/2}$
  - $\cos \beta = r_y'' / (r_y''^2 + r_z''^2)^{1/2}$
  - As $\cos \beta = \cos (-\beta)$, then
    - if $r_z'' > 0$ then the angle must be $(2\pi - \beta) \Rightarrow \beta = -\beta$





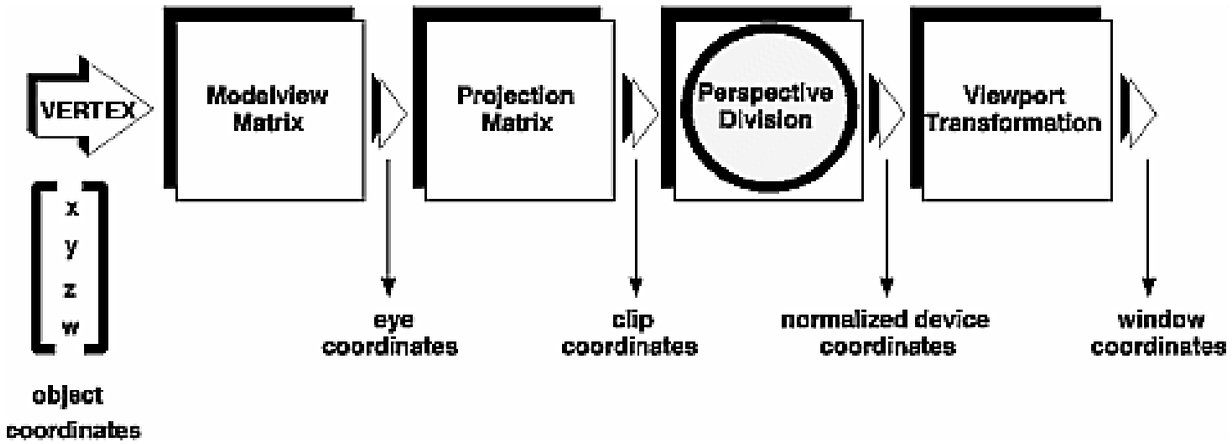
$\beta = \text{acos} (r_y'' / (r_y''^2 + r_z''^2)^{1/2})$
y si $r_z'' > 0 \Rightarrow \beta = -\beta$

Transformations in OpenGL

Content

- Transformations and coordinate systems
- Example of transformation
- Duality of the transformation
- Functions of matrices
- Transformations of modeling and visualization
- Stack of matrices

Transformations and coordinate systems



Example of transformations

```
void reshape(int width, int height) {
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0,
    (GLfloat)height / (GLfloat)width, 1.0, 128.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(0.0, 1.0, 3.0,
            0.0, 0.0, 0.0,
            0.0, 1.0, 0.0);
}
```

Duality of the transformation

- The next two sentences produce the same result:

```
gluLookAt(0.0, 1.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```

```
glTranslatef(0.0, -1.0, -3.0);
```

- It is equivalent to move the camera in one direction than to move the object in the opposite direction.

Modes of the matrix

- **glMatrixMode(*Glenum mode*)**

  GL_MODELVIEW

  GL_PROJECTION

  GL_TEXTURE

Functions of matrices

- **glLoadIdentity**(*void*)
- **glLoadMatrix**{fd}(cont TYPE *m)
- **glMultMatrix**{fd}(cont TYPE *m)

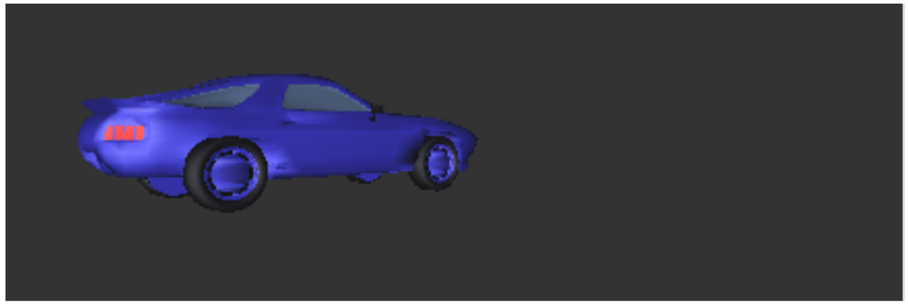
$$M = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

Transformations of modeling and visualization

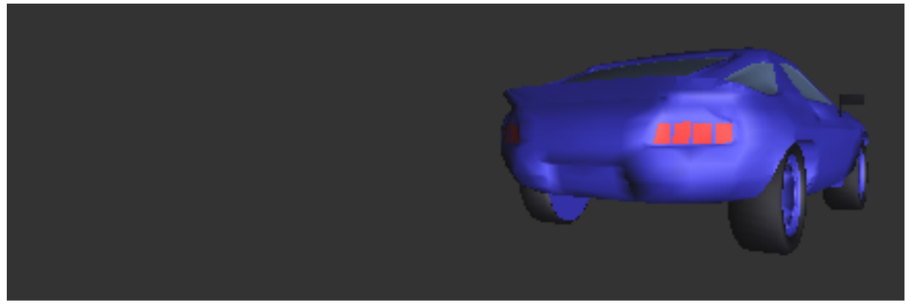
- Both transformations are combined in the same matrix: **`glMatrixMode(GL_MODELVIEW)`**
- They move the coordinates of the object to the coordinates of the camera (eye coordinates)
- Modeling functions:
- **`glTranslate{fd}`**(*`TYPE x, TYPE y, TYPE z`*)
- **`glRotate{fd}`**(*`TYPE angle, TYPE x, TYPE y, TYPE z`*)
- **`glScale{fd}`**(*`TYPE x, TYPE y, TYPE z`*)
- Visualization functions:
- **`gluLookAt()`**

Order of tranformations

- translation(1,0,0)
- rotationY(135º)



```
glRotatef(135.0, 0.0, 1.0, 0.0)
glTranslatef(1.0, 0.0, 0.0)
```



```
glTranslatef(1.0, 0.0, 0.0)
glRotatef(135.0, 0.0, 1.0, 0.0)
```

Order of transformations in OpenGL

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(M1);
glMultMatrixf(M2);
glMultMatrixf(M3);
```

- They apply to the v array:

  M1•M2•M3•v = [M1•[M2•[M3•v]]]

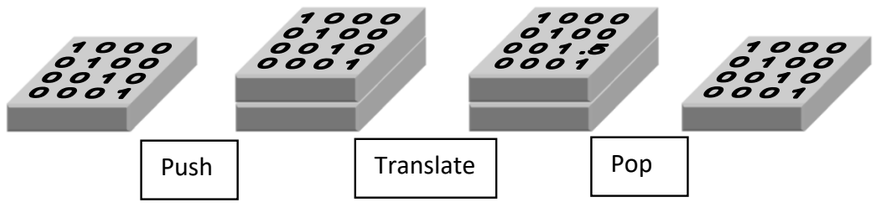
Note: actually, they are applied: [M1•M2•M3]•v

Reference system

- In a global system, the transformations have to be defined from the last to the first.
- In a local system, the transformations have to be defined in the order that they are applied.
- Particular case: turtle of logo

Stack of matrices

- **`glPushMatrix`**(*`void`*)
- **`glPopMatrix`**(*`void`*)



Camera

Content

- Coordinate systems and transformations
- Viewing coordinates
- Coordinate transformation matrix
- Projections
- Window and viewport

Acknowledgments:
To Alex García-Alonso who provided material for these slides (http://www.sc.ehu.es/ccwgamoa/clases)

Projection

- We want to project the 3D space into a plane
- Definition of camera and projection with geometric transformations of the coordinate systems

Coordinate systems and transformations

- Modeling coordinates (local)

  Modeling transformation

- World coordinates (world)

  Viewing transformation

- Viewing coordinates (view)

  Projection transformation

- Device coordinates (screen)

World coordinates

- It unifies the coordinate systems of all the objects of the scene
- Animation is achieved with transformations along time
- Lights and cameras are defined in world coordinates
- The properties of the camera define the viewing coordinates

Viewing coordinates

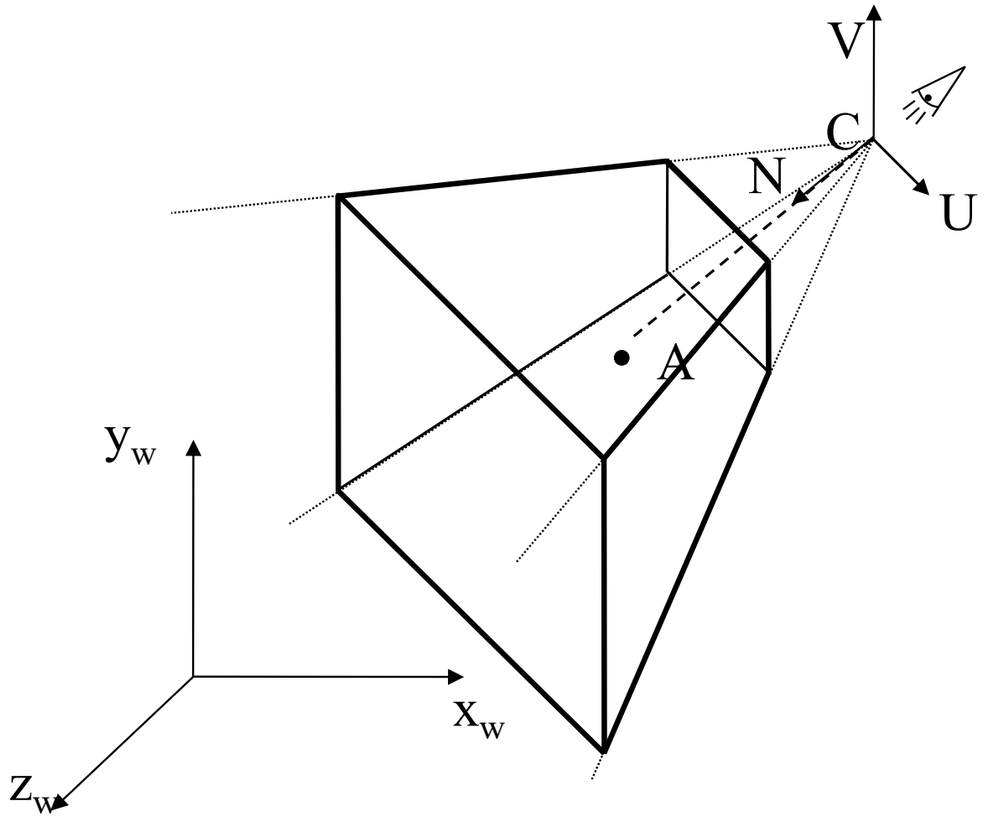
- Camera, eye, view coordinates
- They are the coordinates in the camera system
- They are defined by position and orientation of the camera
- They can include the view volume

Definition of viewing reference frame

- It is defined with the parameters of the camera:
  - View point
  - Direction of viewing
  - View-up vector V
- They define the three dimensional viewing-coordinate frame

Elements of the viewing coordinate frame

- Point C and UVN vectors



- C is the view point
- N is the direction of viewing
- V is the view-up vector (Y axis on the plane)
- U is normal to N and V (X axis on plane)

Rotation transformation

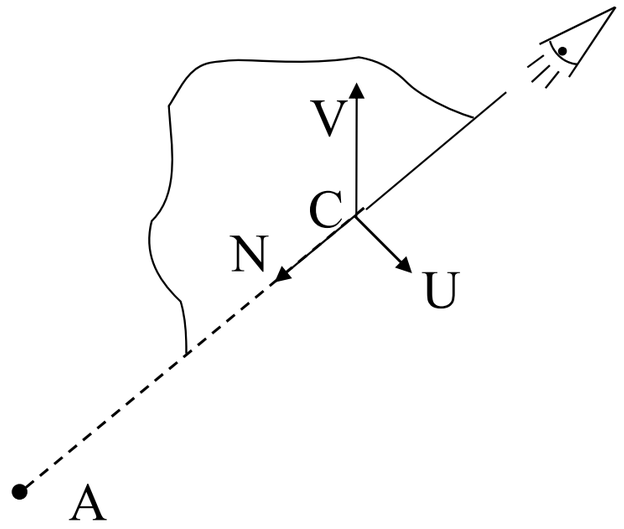
- The transfomation matrix is formed with the unit vectors UVN in world coordinates as rows

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_1, n_2, n_3)$$

$$\mathbf{u} = \frac{\mathbf{V} \times \mathbf{N}}{|\mathbf{V} \times \mathbf{N}|} = (u_1, u_2, u_3)$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_1, v_2, v_3)$$

Hearn & Baker, 12-2



Transformation matrix to viewing coordinates

- Composition of translation and rotation
- $T_{view} = R \bullet T$
- It is a left-handed system (X axis to the left)

$$T = \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Types of projections

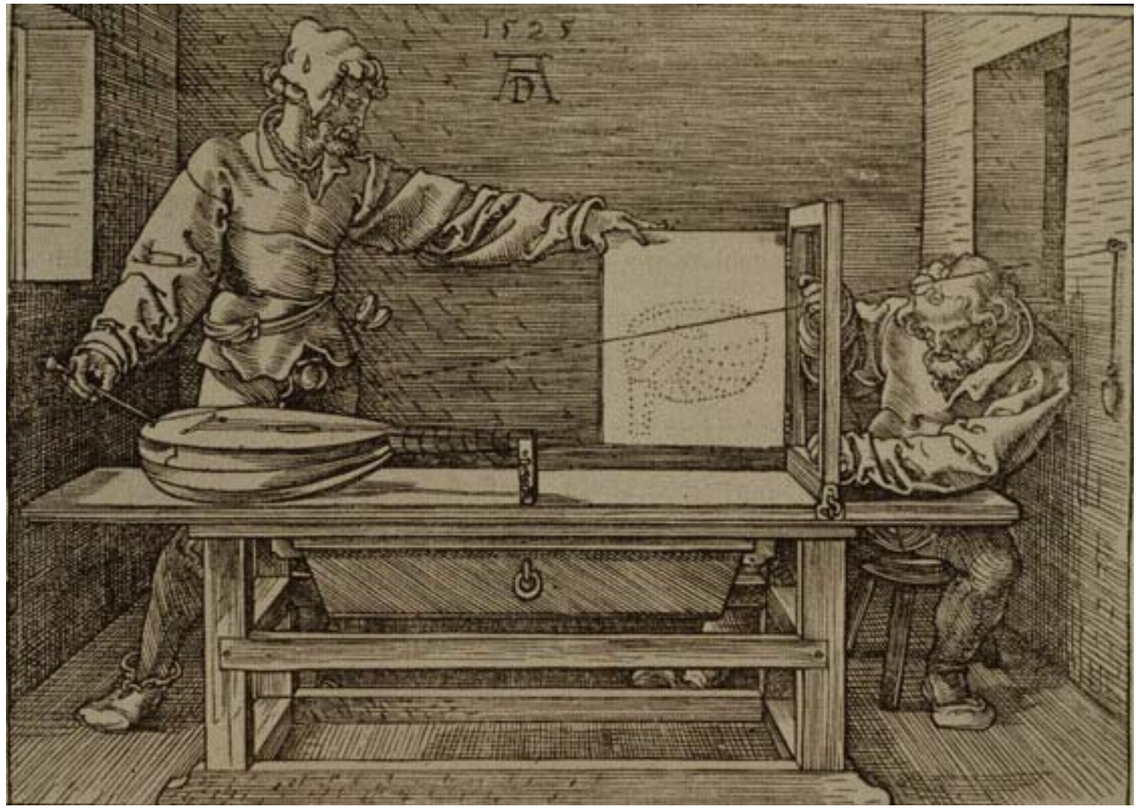
- Parallel projection
  - orthogonal
  - oblique (projection not perpendicular to the view plane)
- Perspective projection

Parallel projection

- Orthogonal projection in view coordinates: the z coordinate is eliminated

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective projection



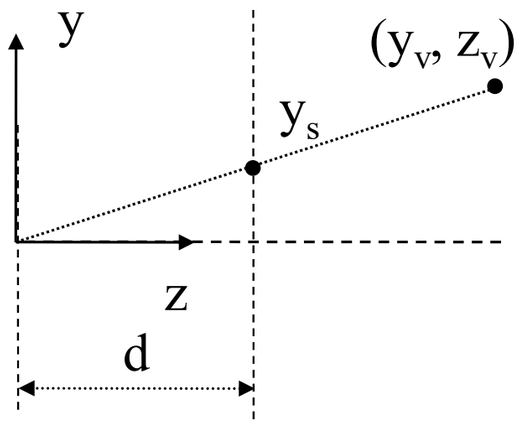
Man Drawing a Lute, Woodcut, 1525, Albrecht Dürer.

http://www.usc.edu/schools/annenberg/asc/projects/comm544/library/images/626.jpg

Features of perspective projection

- More real: it is the projection that happens in the eye and in a camera
- Parallel lines in the scene converge in a vanishing point
- The quantity of vanishing points is defined by the quantity of parallel lines that intersect with the projection plane

Transformations of the perspective projection



$$\frac{y_v}{z_v} = \frac{y_s}{d} \Rightarrow y_s = \frac{y_v}{z_v / d}$$

With a matrix expression:

$$\begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix} \circ \begin{bmatrix} x_v \\ y_v \\ z_v \\ 1 \end{bmatrix}$$

$$x_s = \frac{X}{w}$$

$$y_s = \frac{Y}{w}$$

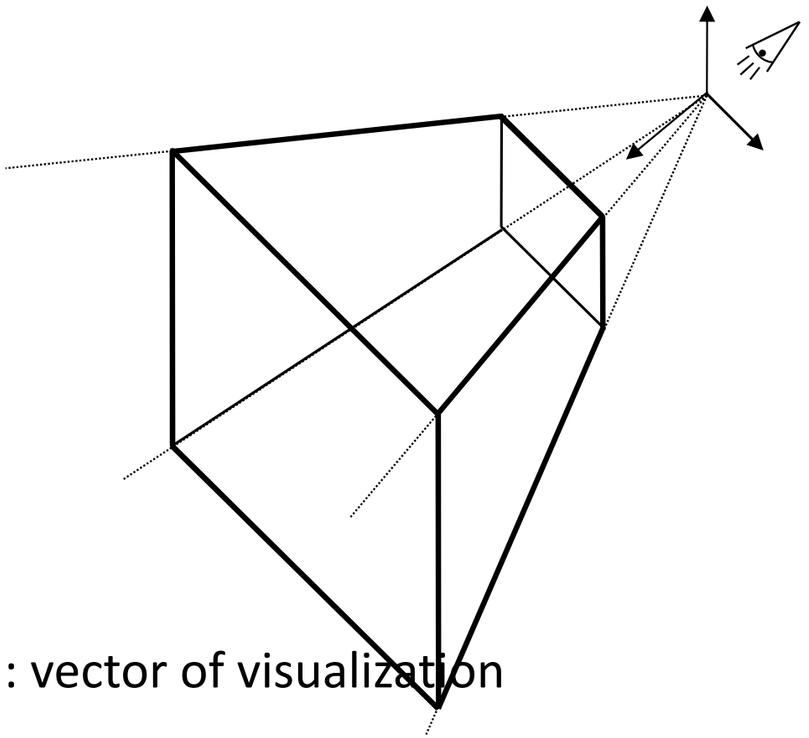
$$z_s = \frac{Z}{w} = d$$

Other issues

- Visualization volume
  - Sides of the pyramid
  - Near plane and far plane (near and far)

$$x_v = \frac{h_x z_v}{d} \qquad y_v = \frac{h_y z_v}{d}$$

$$z_v = n \qquad z_v = f$$



- Hide back sides
  - Np: normal of the polygon, N: vector of visualization

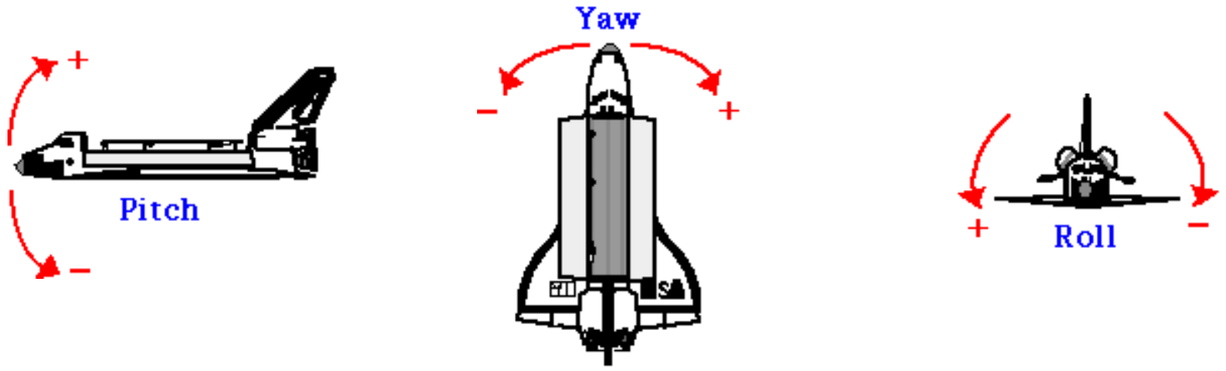
$$N_p \bullet N > 0$$

Camera movements

- Of the camera position
  - Around the camera axis
  - Around the scene axis
- Of the point of attention
- Simultaneous of both
- Object in hand
- Walking and flying

Airplane analogy

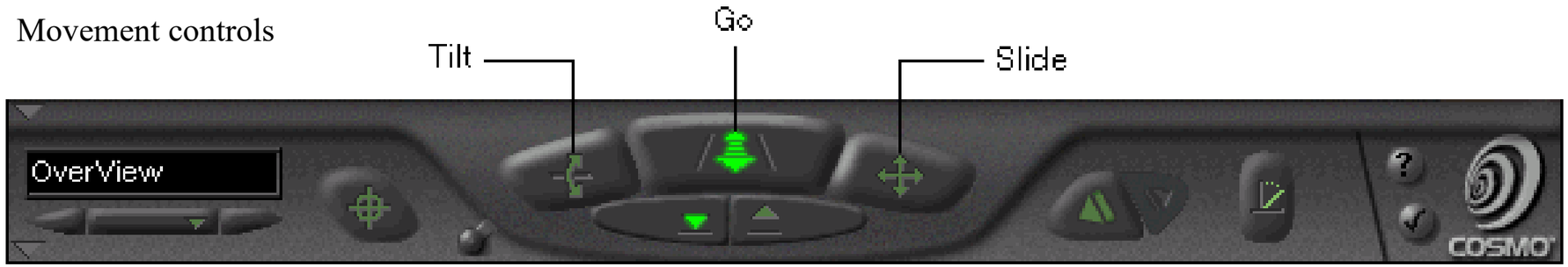
- Rotation around X: Pitch (cabeceo)
- Rotation around Y: Yaw (giro)
- Rotation around Z: Roll (balanceo)



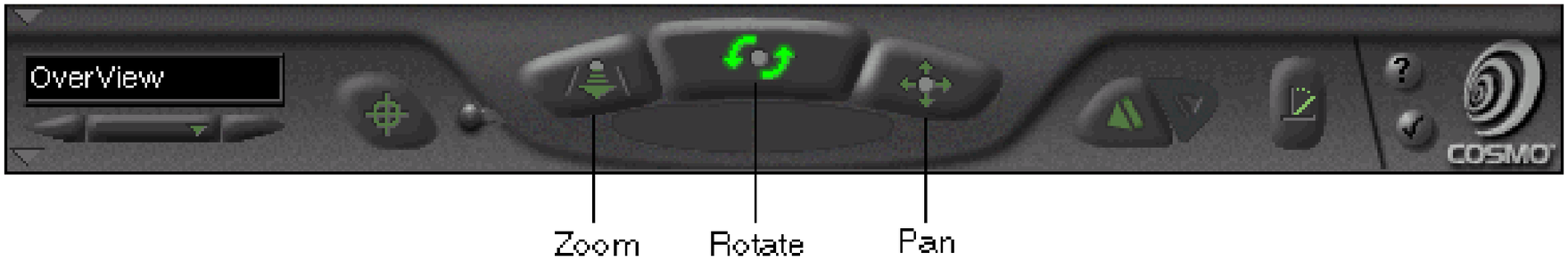
http://liftoff.msfc.nasa.gov/academy/rocket_sci/shuttle/attitude/pyr.html

Cosmoplayer controls

Movement controls



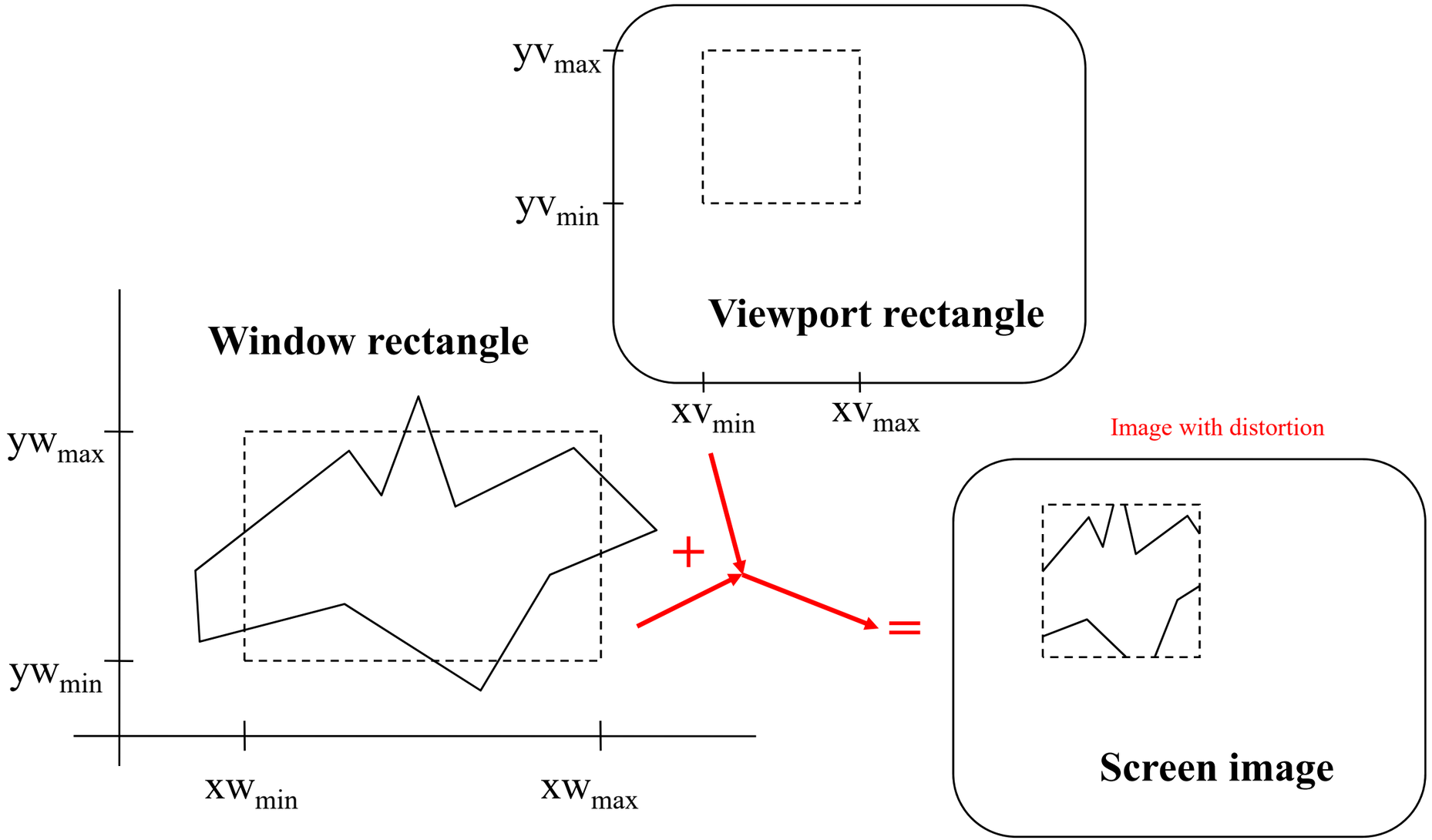
Examine controls



Windows of presentation

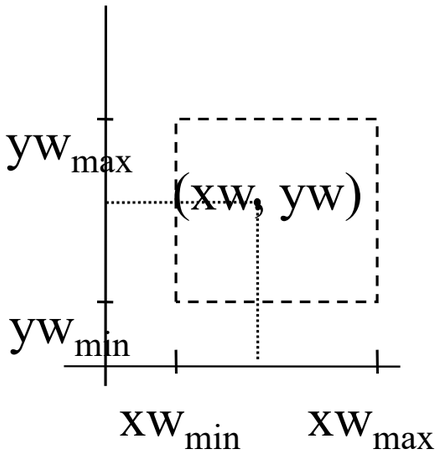
- Object Window
- The projection of the camera create 2 dimension coordinates
- The device coordinates are independent of the scene coordinates
- It is necessary to transfer from window coordinates to the device coordinates

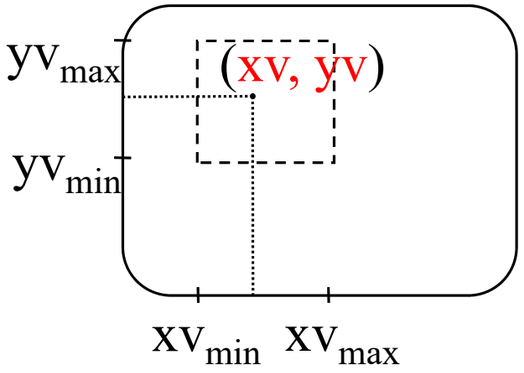
Window and viewport



Transformation to viewport

- To calculate the coordinates in viewport (xv, yv) of a point in window coordinates (xw, yw) (previously (x_s, y_s))
- The existing relations are:





$$\frac{xv - xv_{\min}}{xv_{\max} - xv_{\min}} = \frac{xw - xw_{\min}}{xw_{\max} - xw_{\min}}$$

$$\frac{yv - yv_{\min}}{yv_{\max} - yv_{\min}} = \frac{yw - yw_{\min}}{yw_{\max} - yw_{\min}}$$

Issues about transformations

- Distortion, it is caused by the different rate of window and viewport
  - allow
  - Avoid through change in window or viewport
- Clipping
  - Cutting the segments and polygons that intersect the window

Lighting

Content

- Factors
- Reflection
- Ambient Light
- Diffuse Light
- Specular Light
- Illumination model

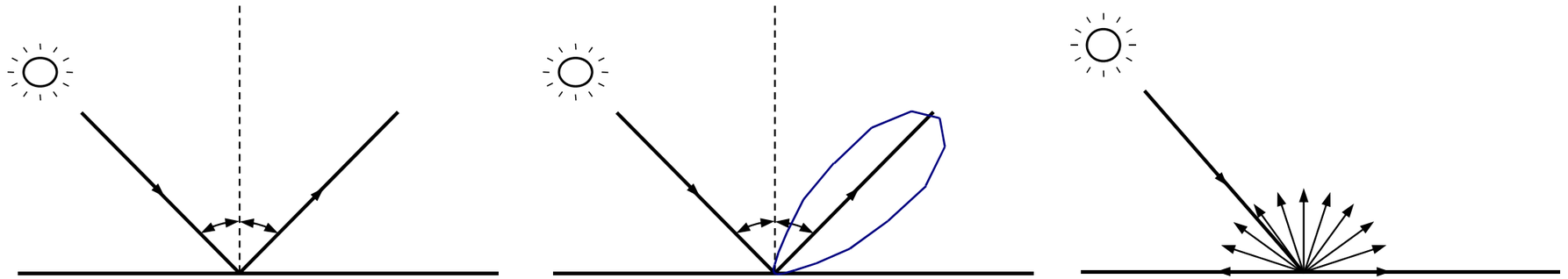
Acknowledgments:
To Alex García-Alonso for the material for this slides
(http://www.sc.ehu.es/ccwgamoa/clases)

Factors of illumination

- We perceive images made by light
- The light we see on a point depends on:
  - the light that arrives from the light source
  - the orientation of the point
  - the features of the material
  - the light that arrives from other points
  - the position of the observer

Reflection of the light

- The light can be reflected:
  - perfect specular reflection
  - imperfect specular reflection
  - diffuse



Illumination model

- The reflected light of a surface is defined by:

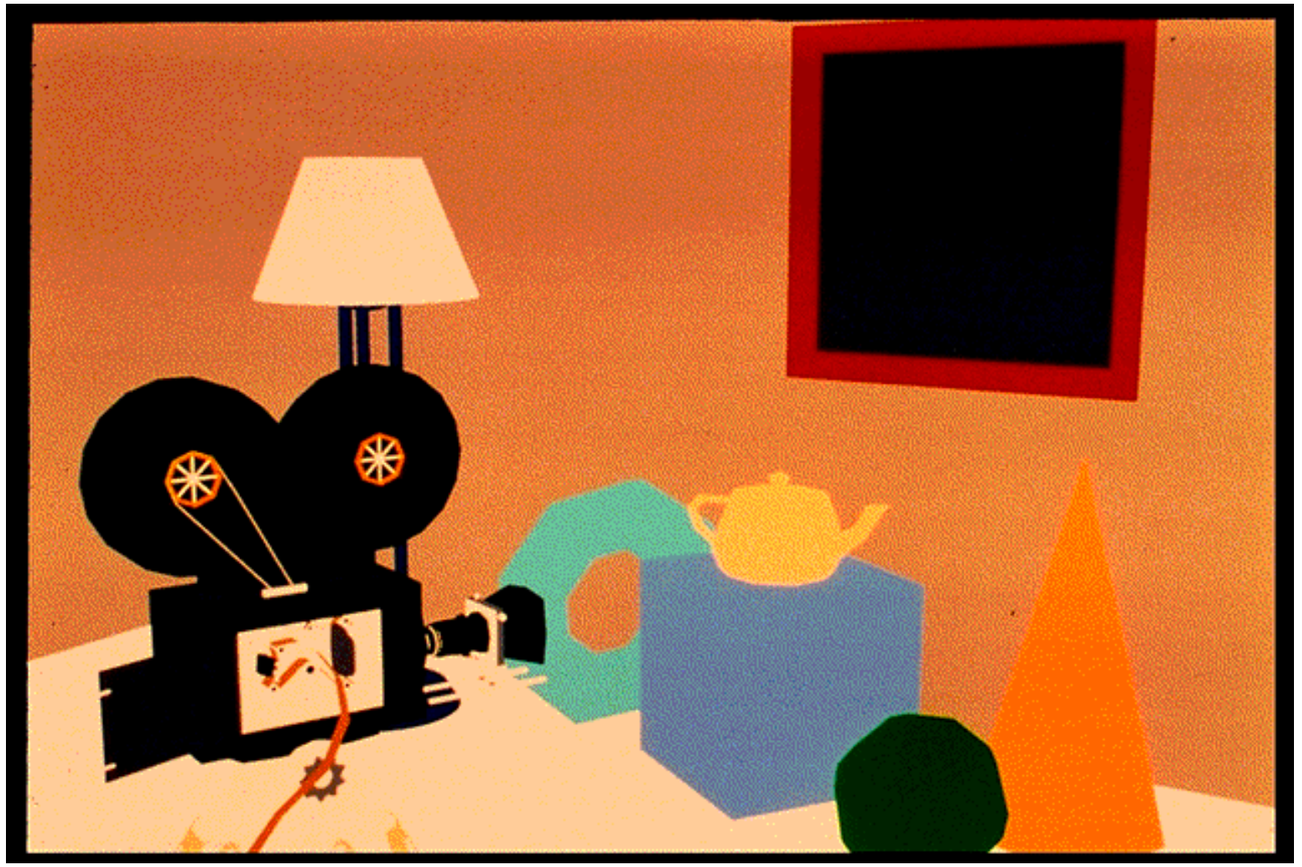
  Ambient light + Diffuse reflection + Specular reflection

$$I = k_a I_a + k_d I_d + k_s I_s$$

Ambient light

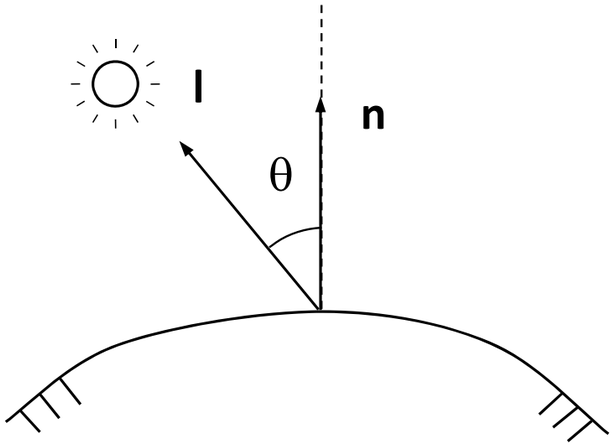
- $k_a I_a$
- I_a is the ambient light of the scene
- k_a is a property of the material
- It is a simplification of the global illumination model
- It substitutes the lights that doesn’t arrive directly from the source lights

Ambient light - image

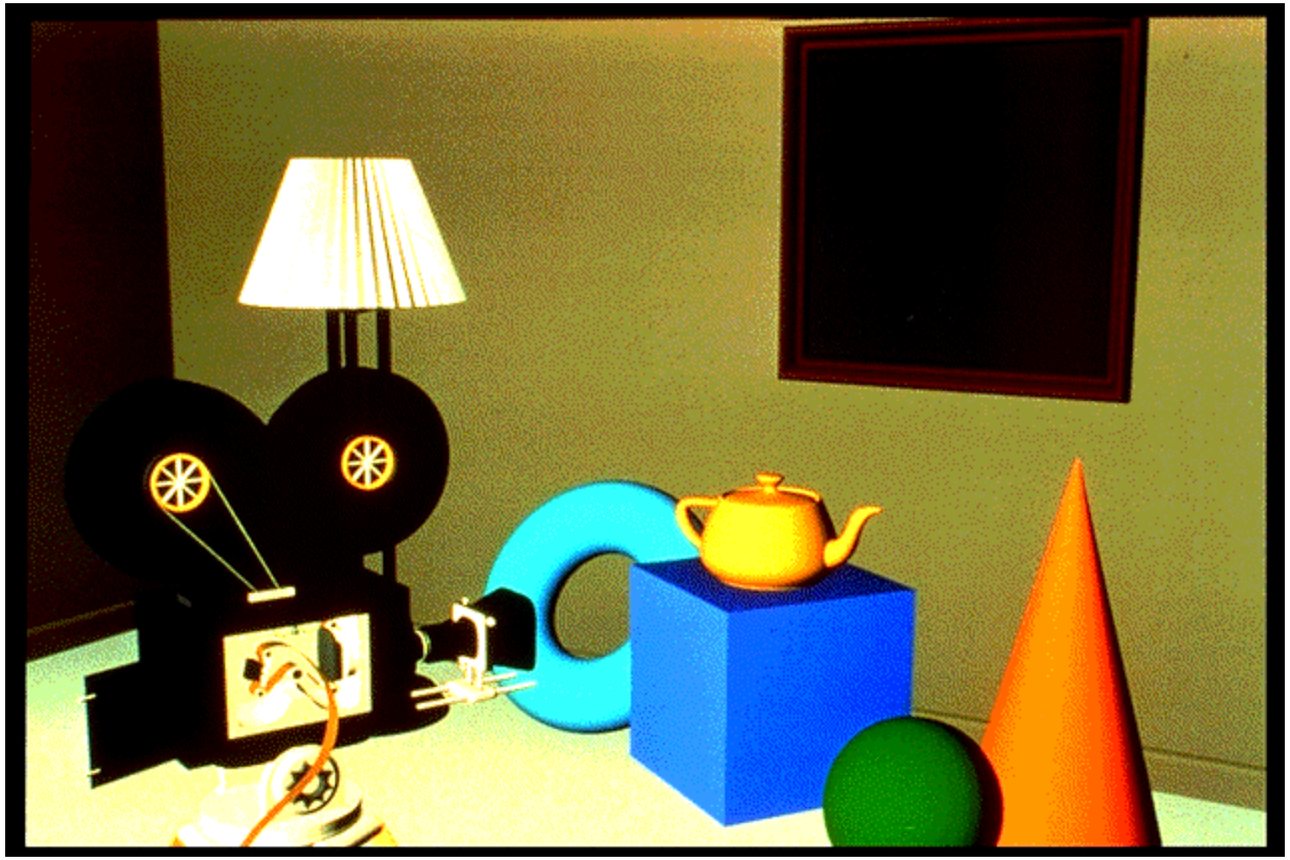


Diffuse reflection

- Lambert's cosin Law
- $k_d I_d = k_d I_p \cos \theta =$
  $= k_d I_p (n \bullet l)$
- I_p is the light from a point source light
- k_d is a property of the material
- It is independent of the observer
- It is the main component of the object's color



Diffuse reflection - image



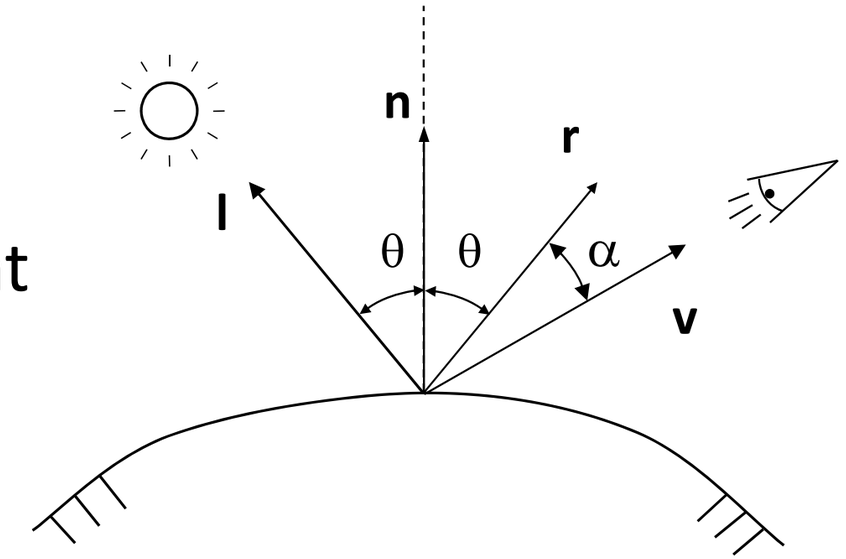
Specular reflection

- The perfect specular reflection only produces reflection on a point
- The imperfect specular reflection is used in CG
- The intensity decreases when we move away from the reflected ray direction
- Fresnel Law: $\cos^s \alpha$
- It produces the shine of the objects

http://www.forestal.uchile.cl/centro/frutillar/fotozona/2puesta%20sol%20clara.jpg

Specular reflection - calculation

- $k_s I_s = k_s I_p \cos^s \alpha =$
  $= k_s I_p (r \bullet v)^s$
- I_p is the light of a point light source
- k_s is a property of the material
- s defines the diffusion, it is the specular parameter



Reflected vector

- Calculation of the reflected vector r

$r = r_1 + r_2$ [1]

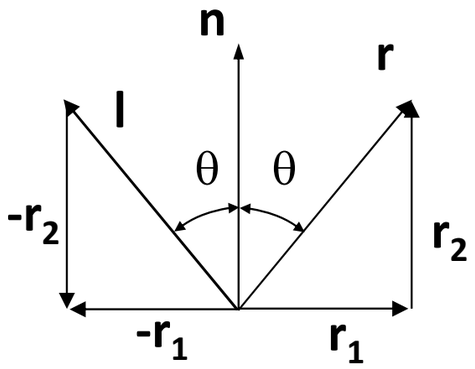
$- r_1 = l + (-r_2) \Rightarrow r_1 = - l + r_2$ [2]

Replace in [1]:

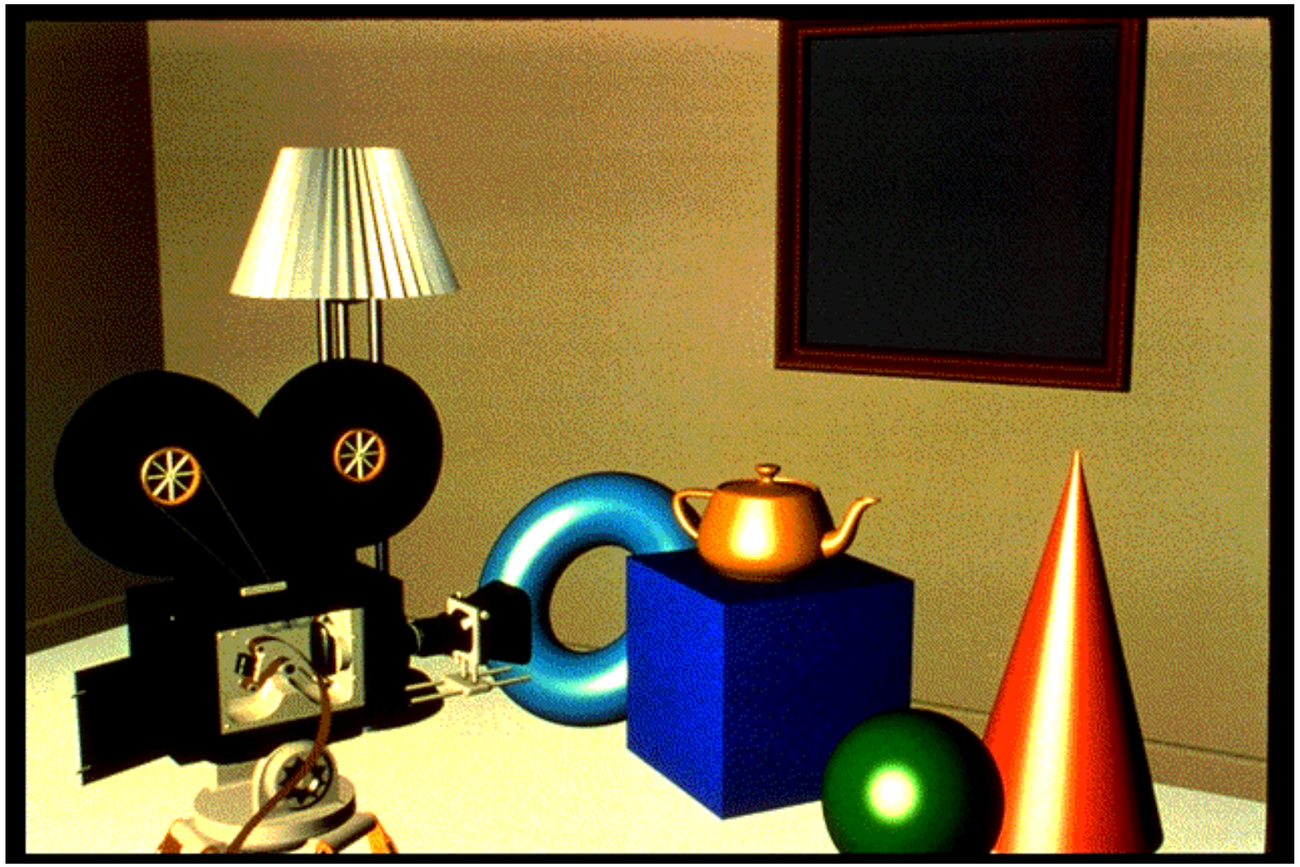
$r = - l + r_2 + r_2 = 2r_2 - l$ [3]

$r_2 = (n \bullet l)\, n$ [4]

Replace in [4]: $r = 2\,(n \bullet l)\, n - l$ [5]



Specular reflection - image

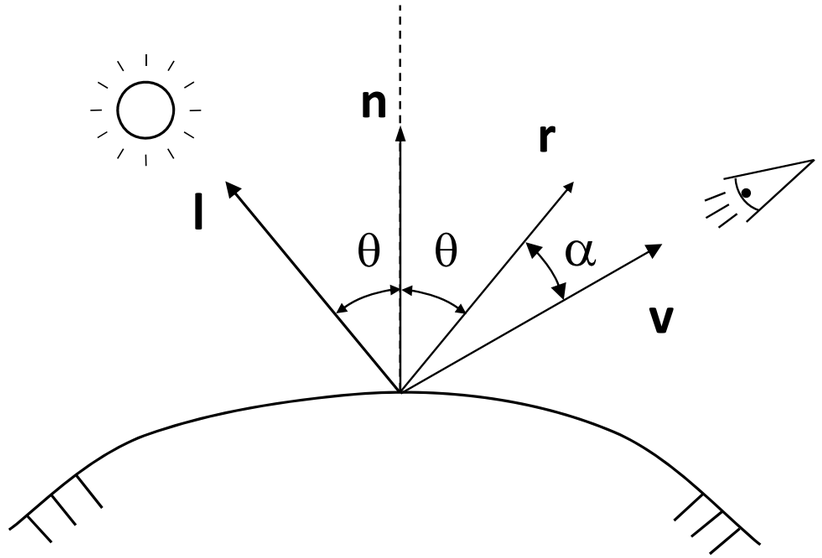


Illumination Model

- The light reflected by a surface:

  Ambient light + Diffuse reflection + Specular reflection

$$I = k_a I_a + k_d I_p (n \bullet l) + k_s I_p (r \bullet v)^{s}$$



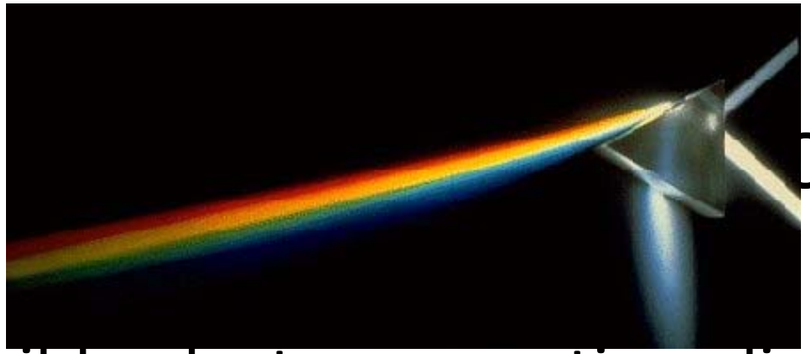
Render

- Gouraud render
- Phong render
- Global illumination Model
  - ray tracing
  - radiosity

Color

Contents

- Light and color
- The visible light spectrum
- Primary and secondary colors
- Color spaces
  - RGB, CMY, YIQ, HLS, CIE
  - CIE XYZ, CIE xyY and CIE diagram properties



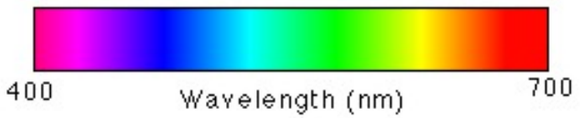
o we see?

- Visible electromagnetic radiation
- The type of electromagnetic wave that is visible to the human eye.
- Electromagnetic radiation that has a wavelength in the range from about 4,000 (violet) to about 7,700 (red) angstroms and may be perceived by the normal unaided human eye.

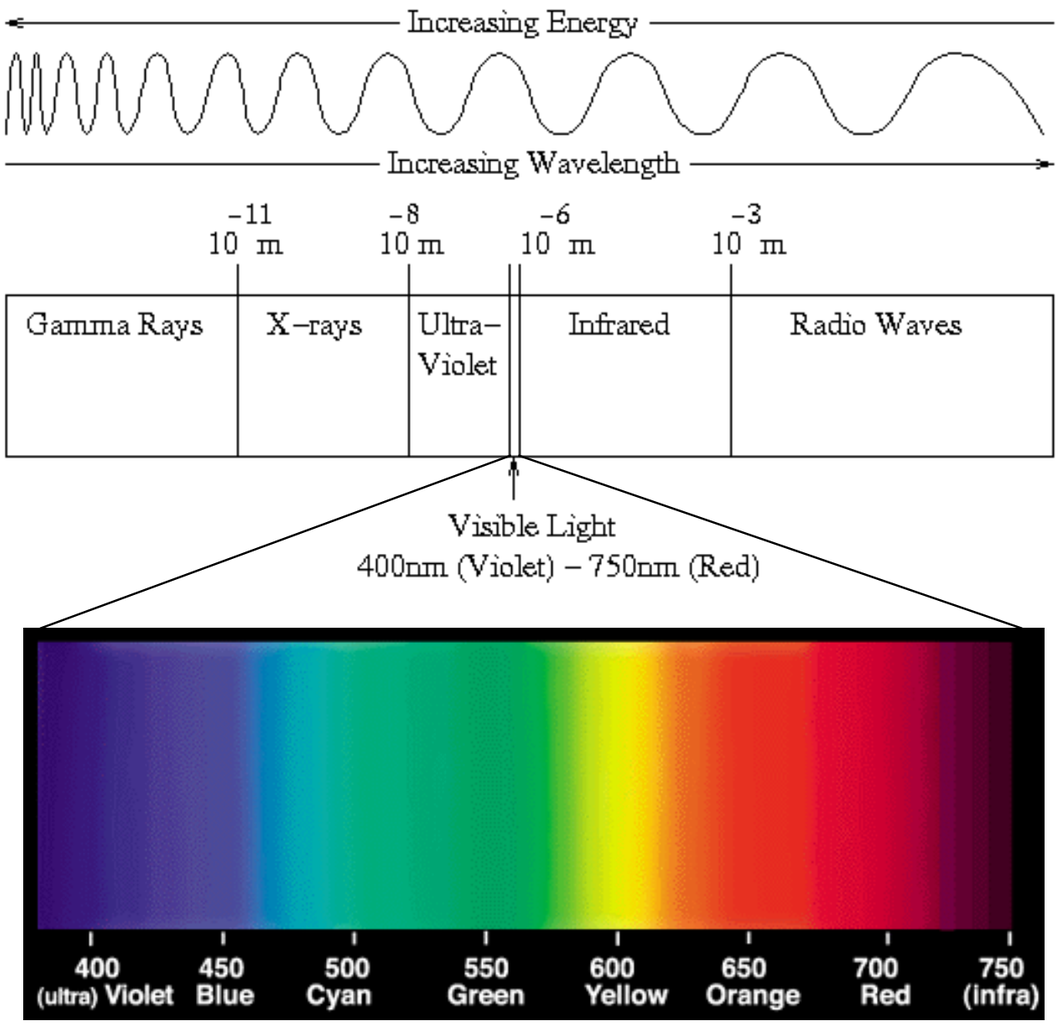
http://www.answers.com/color

Electromagnetic radiation

- Electromagnetic radiation is a kind of energy radiated in the form of a wave
- The frequency define the energy
  - $E = h \nu$
  - And the hue: $\lambda = c / \nu$

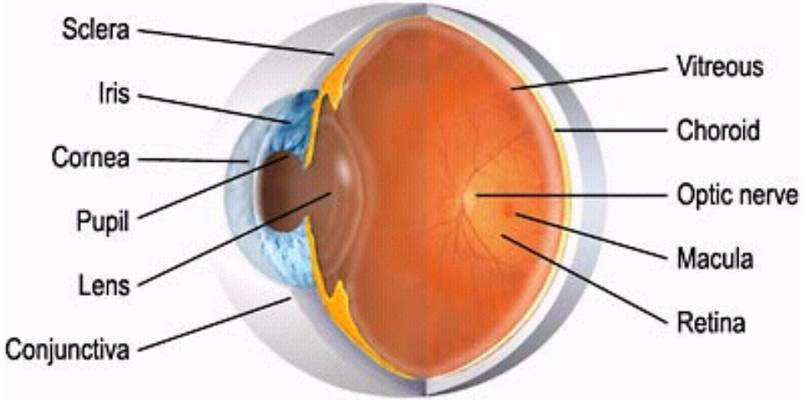


Electromagnetic spectrum and the visible light spectrum

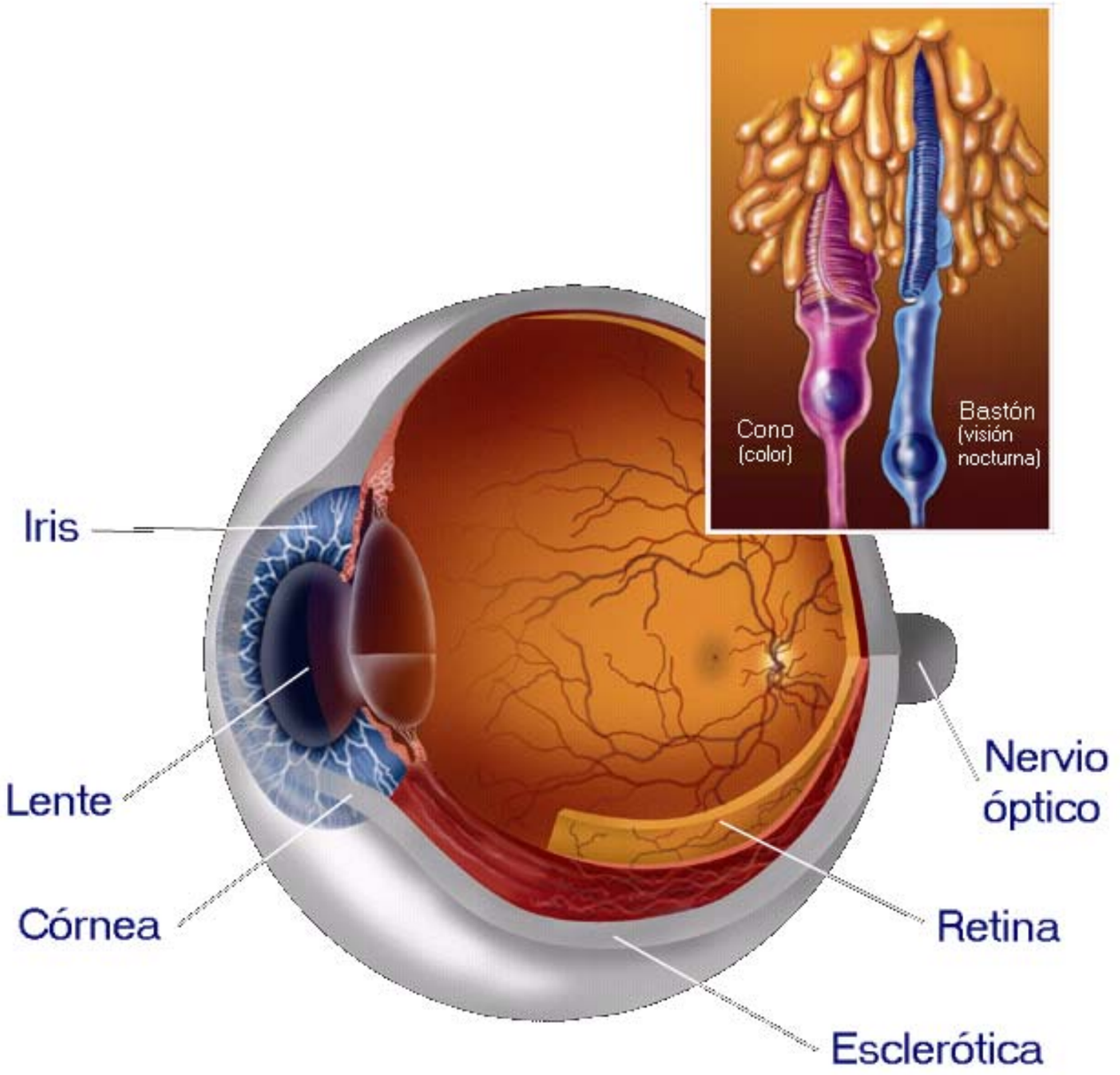


What is color?

- It is a sensation
- “Color is actually light waves that hit our eyes, translated into nerve impulses, and interpreted by our brains as all the various colors around us.”
- Three components: RGB
- Cones
  - Reds (low frequency)
  - Greens (medium frequency)
  - Blues (high frequency)



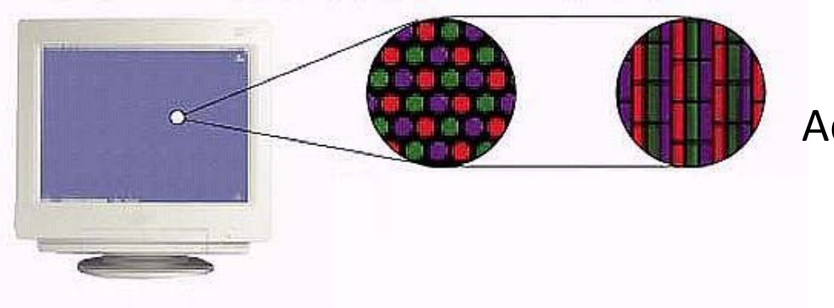
http://www.stlukeseye.com/images/illustrations/anatomy_globe.jpg



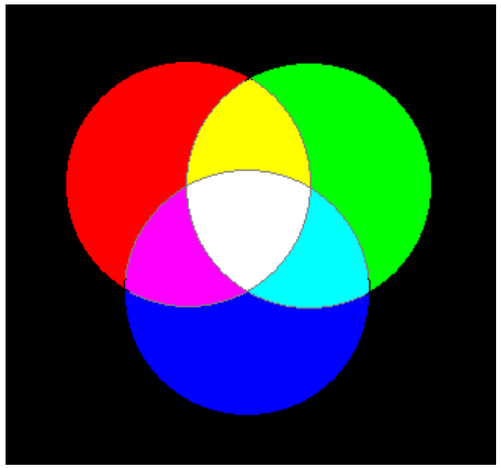
http://www.uic.edu/com/eye/department/support/images/AdrienneVersion.jpg

Mixing colors

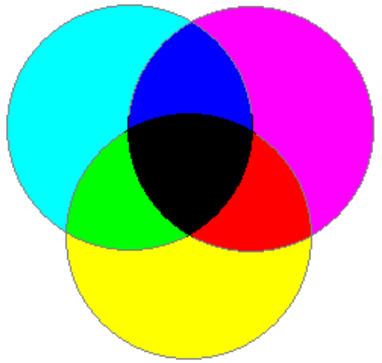
Primary and secondary colors



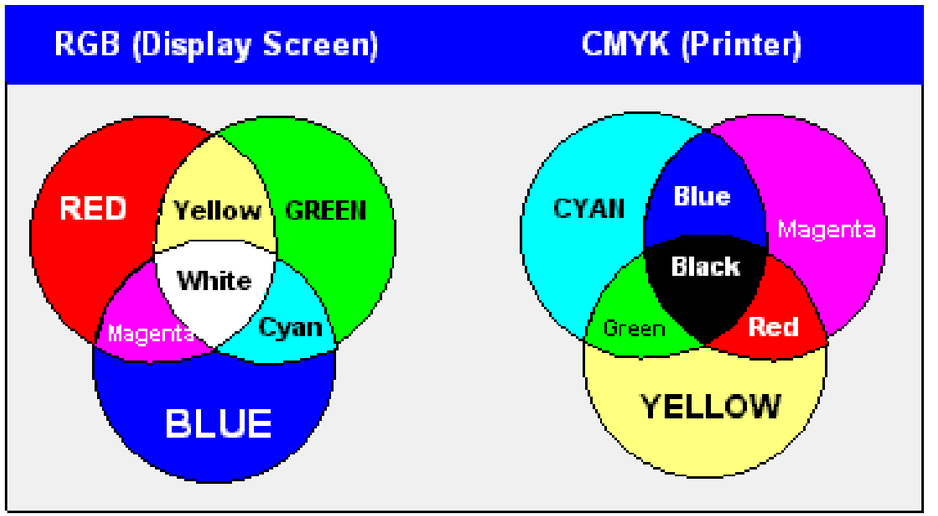
Additive Colors



Subtractive Colors



From Computer Desktop Encyclopedia

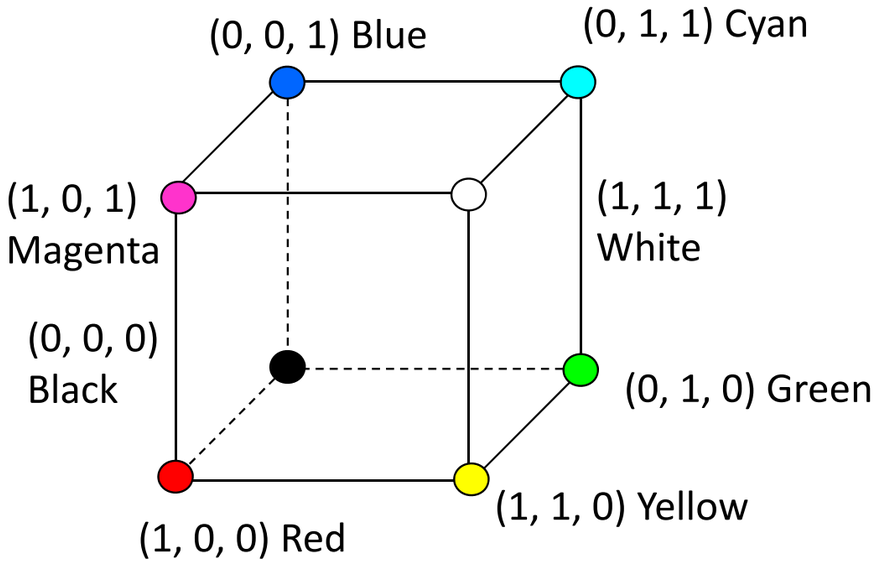


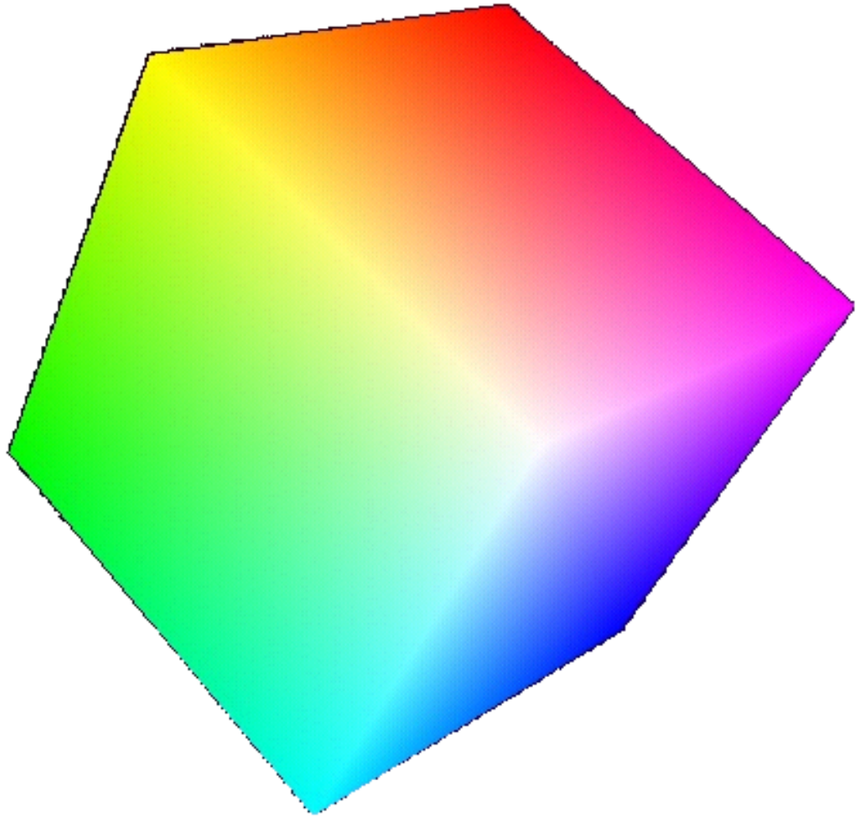
Color spaces

- Systems to define colors numerically
- Device oriented
  - RGB
  - CMY
  - YIQ
- Human oriented
  - HSV
  - CIE

RGB

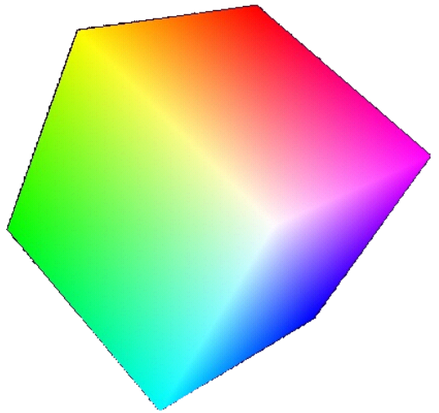
- Additive model
  - Intensity of the three components





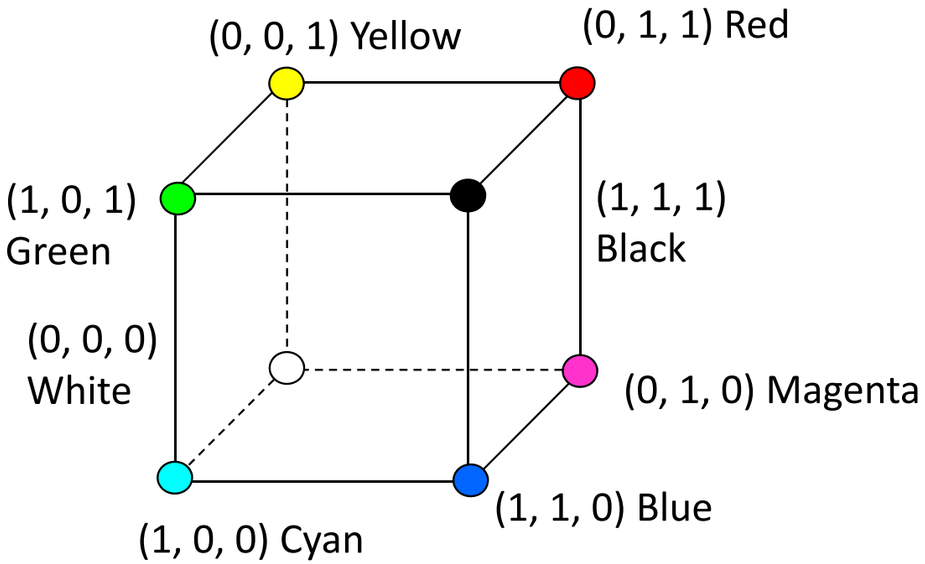
RGB

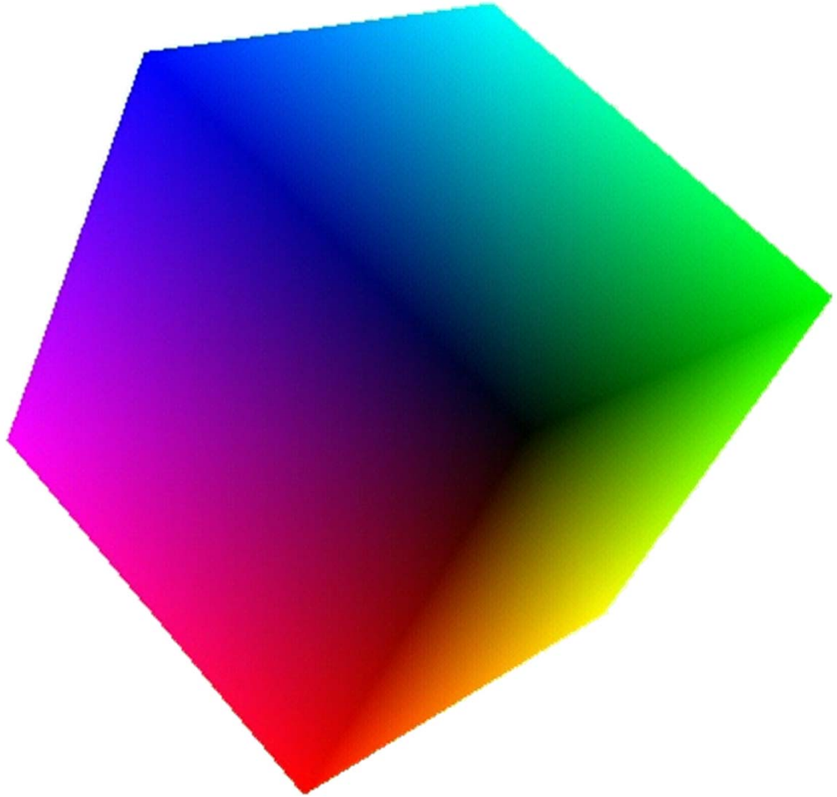
- Features
  - It is used by hardware devices
    - True color: 256^3 = 16.777.216
  - It is a standard on computing
    - bgcolor="#FFFFFF”
- Problems
  - Their variations are not lineal
  - It isn’t intuitive



CMY

- Subtractive model

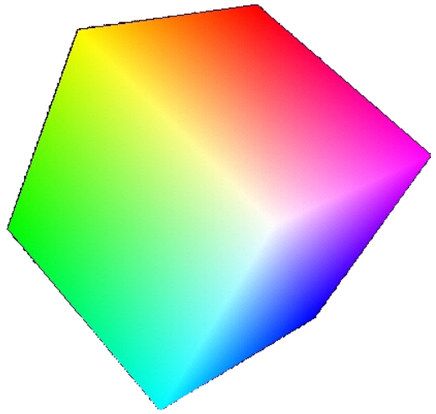


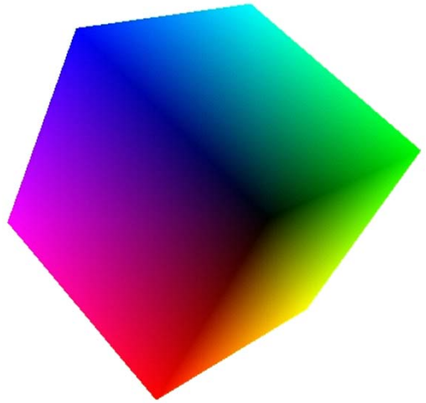


CMY

- Complementary of RGB

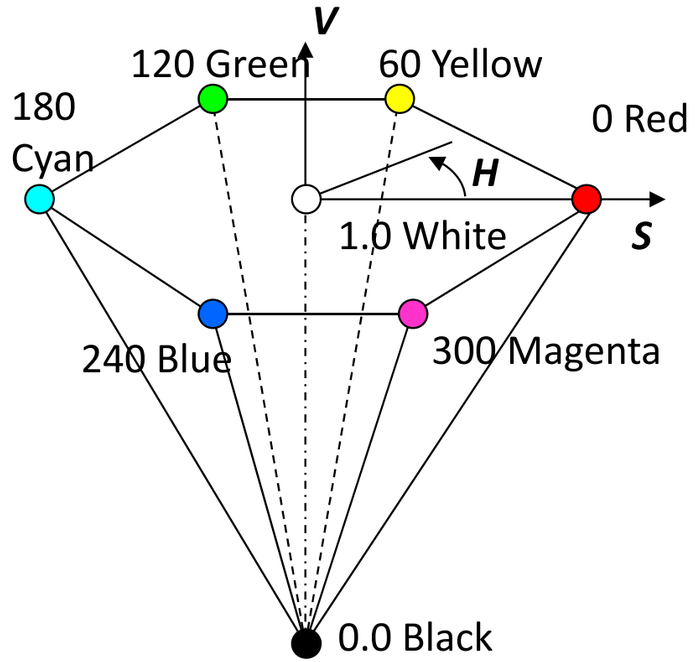
  [CMY] = [1,1,1] - [RGB]
- Used in printing
- Use of CMYK
  - Black (K) is added

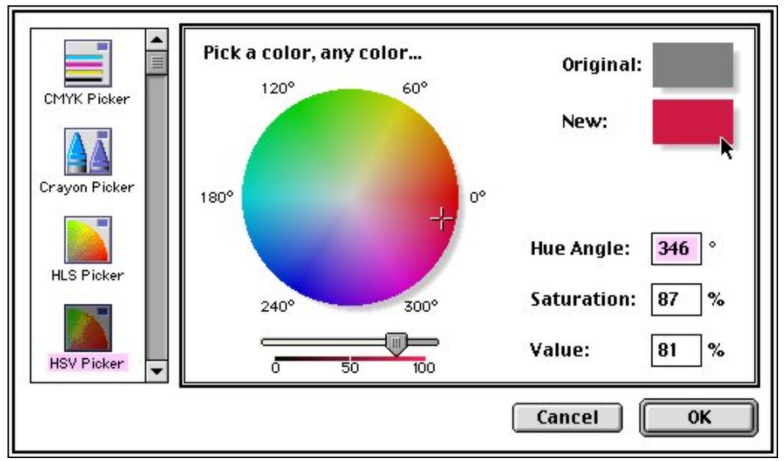




HSV

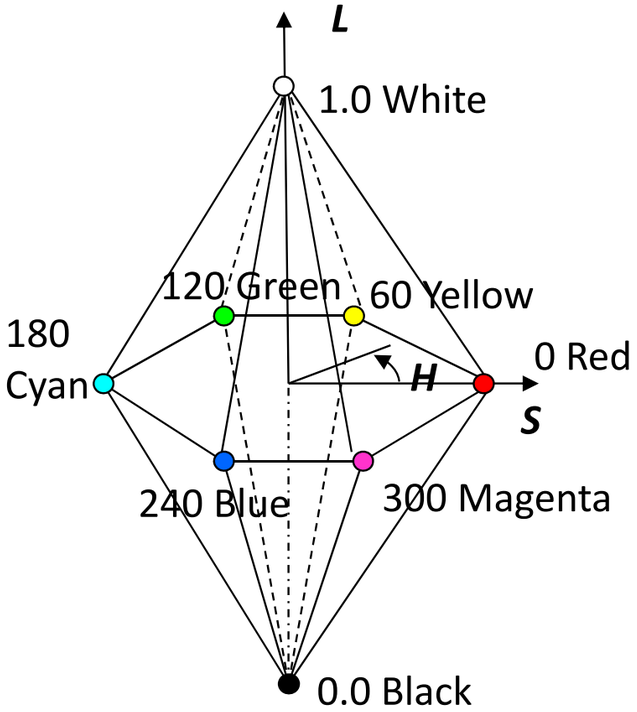
- Projection of RGB diagram about the biggest diagonal
- Polar coordinates
  - Hue
  - Saturation
  - Value or brightness
- Transformation to RGB (no lineal)

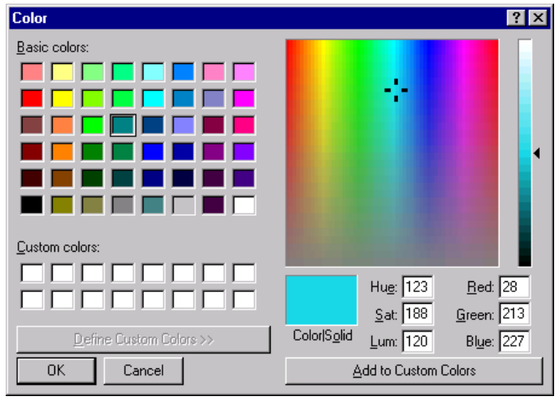


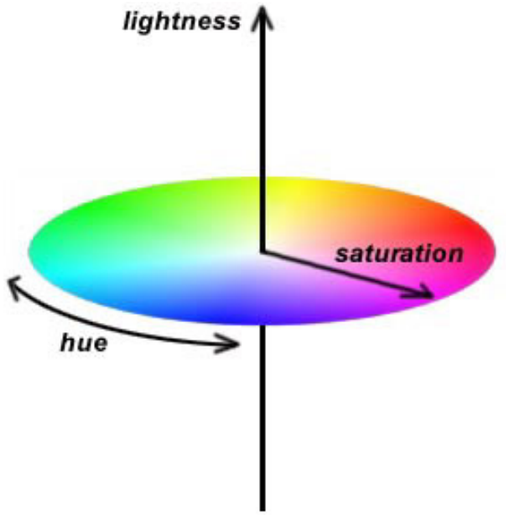


HLS

- Two cones joined, with white on the upper vertex
- Variation of HSV
  - L: lightness





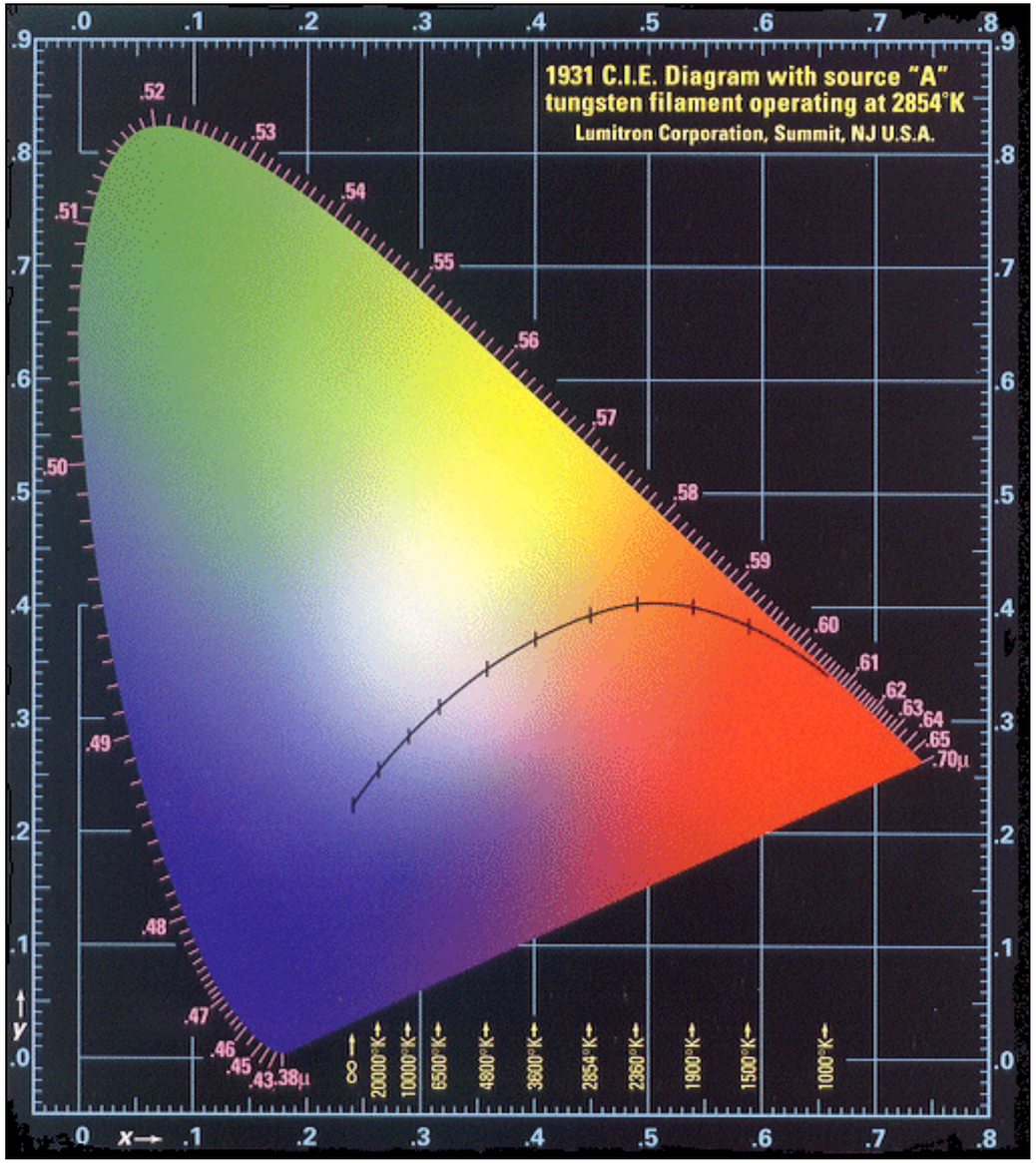


YIQ

- Lineal transformation of RGB
- Used by the NTSC (National Television Standards Committee)
- Y is brightness (used in black-and-white television monitors)

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

CIE Diagram



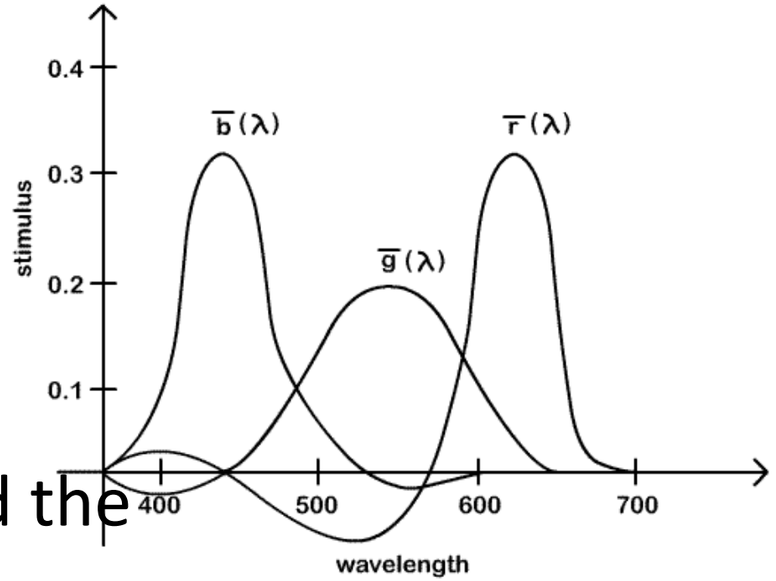
http://www.lumitroncorp.com/rsrcs/CIE.gif

CIE

- 1931: measurement of color by the CIE (Commission International de L'Eclairage)
- Based on Tristimulus Vision Theory
- Represents any color detected by the human being

Definition of CIE diagram

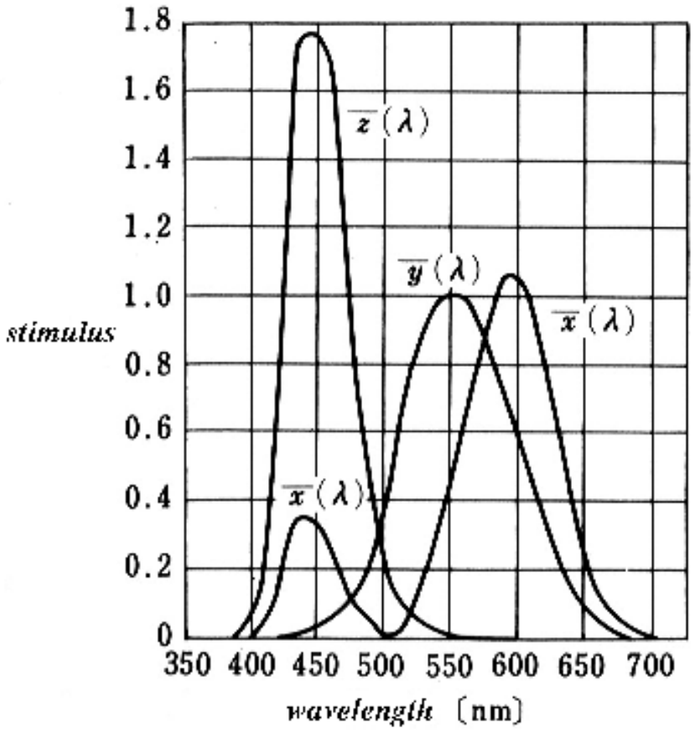
- Experimentally
  - Red (700 nm)
  - Green (546,1 nm)
  - Blue (435,8 nm)
- There is negative values (added the primary to the sample)
  - They can’t be used
  - It is made adding the primary to the sample



http://www.imel1.kuis.kyoto-u.ac.jp/education/dip-arch/pre/images/graph10.gif

The three standard primaries of CIE

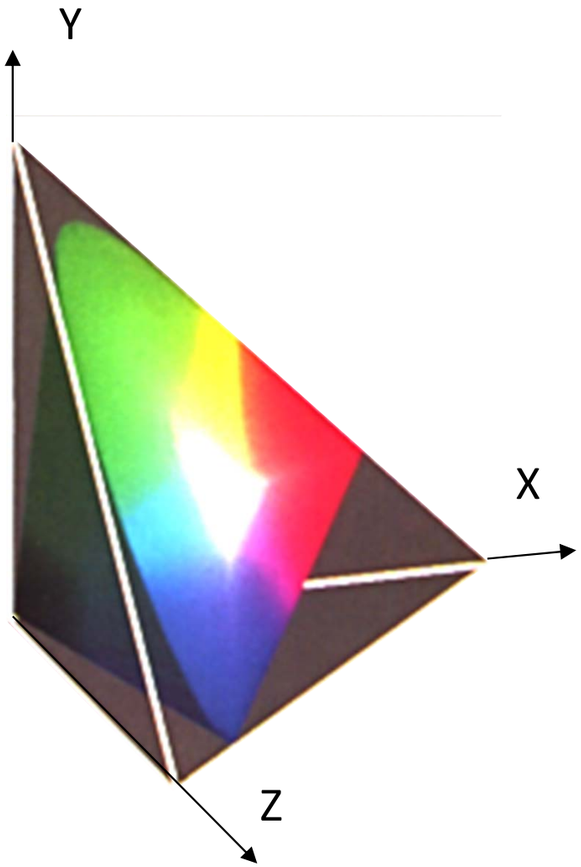
- They are not real colors
- The functions are defined in tabular form (at 1 nm interval)
- They are standards to define colors



http://www.imel1.kuis.kyoto-u.ac.jp/education/dip-arch/pre/images/xyz.gif

CIE XYZ

- One color is defined by the X, Y, Z components.
- The projection on the XY plane produces the CIE Chromatic Diagram.

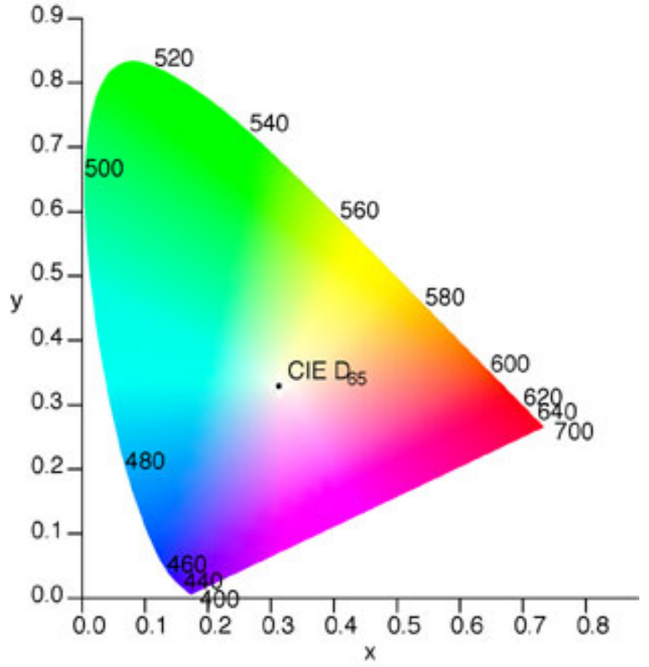


CIE xyY

- Normalizing the values:

  $x + y + z = 1$

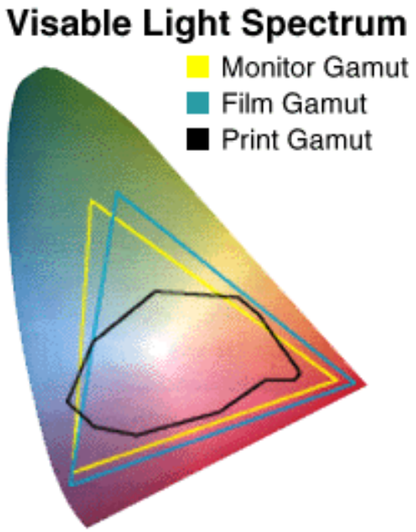
- Y is the luminance information
  - function $y(\lambda)$ was chosen in this way



http://hypertextbook.com/physics/waves/color/chromaticity.jpg

Properties of CIE diagram

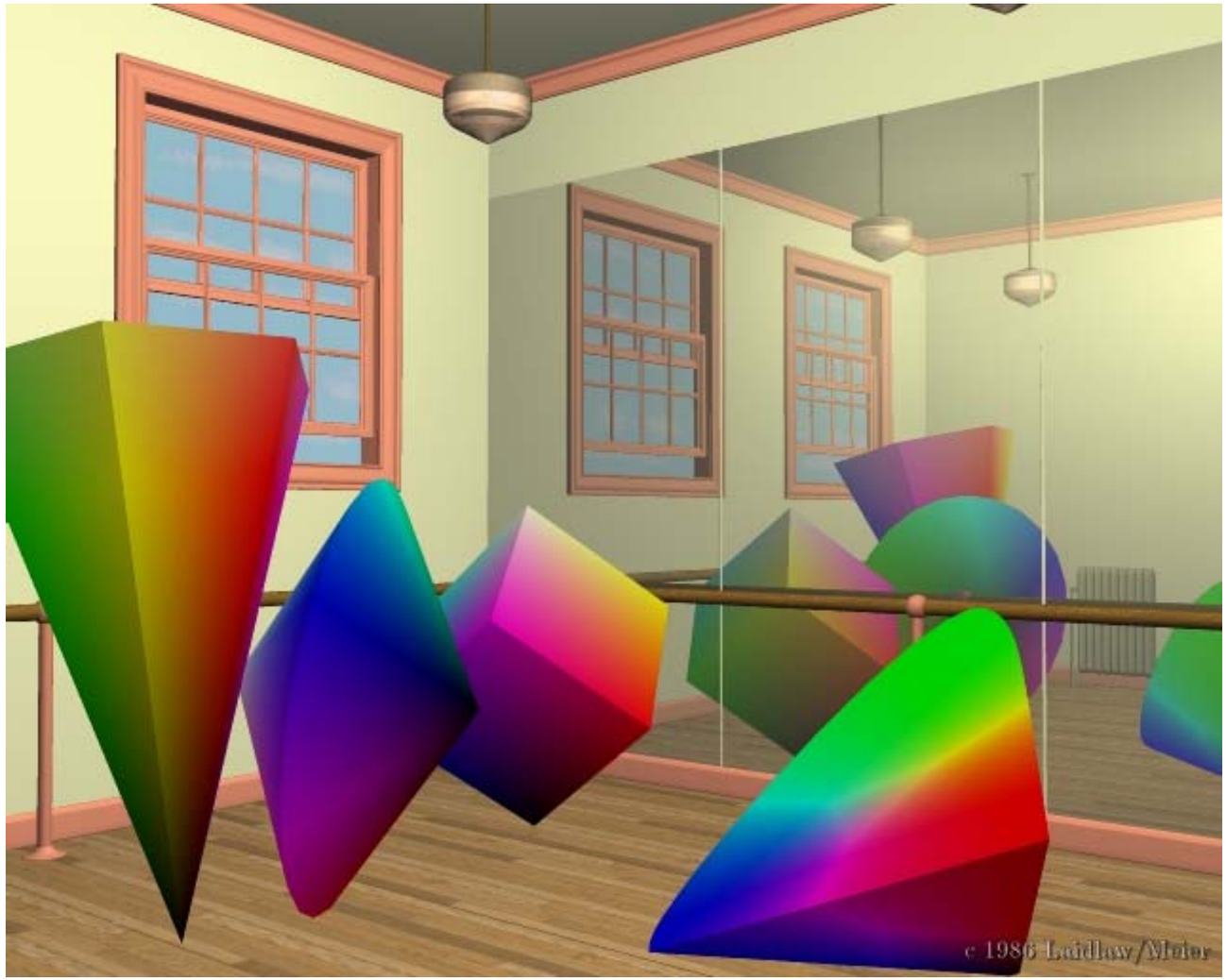
- Gamut of different devices
- The periphery is the spectrum colors
- The purple line are non-spectral colors
- The spectral component of a color is calculated with the line to the center



Color gamuts of color monitor, film and print shown transposed on the 1931 CIE x*y*z* color space.

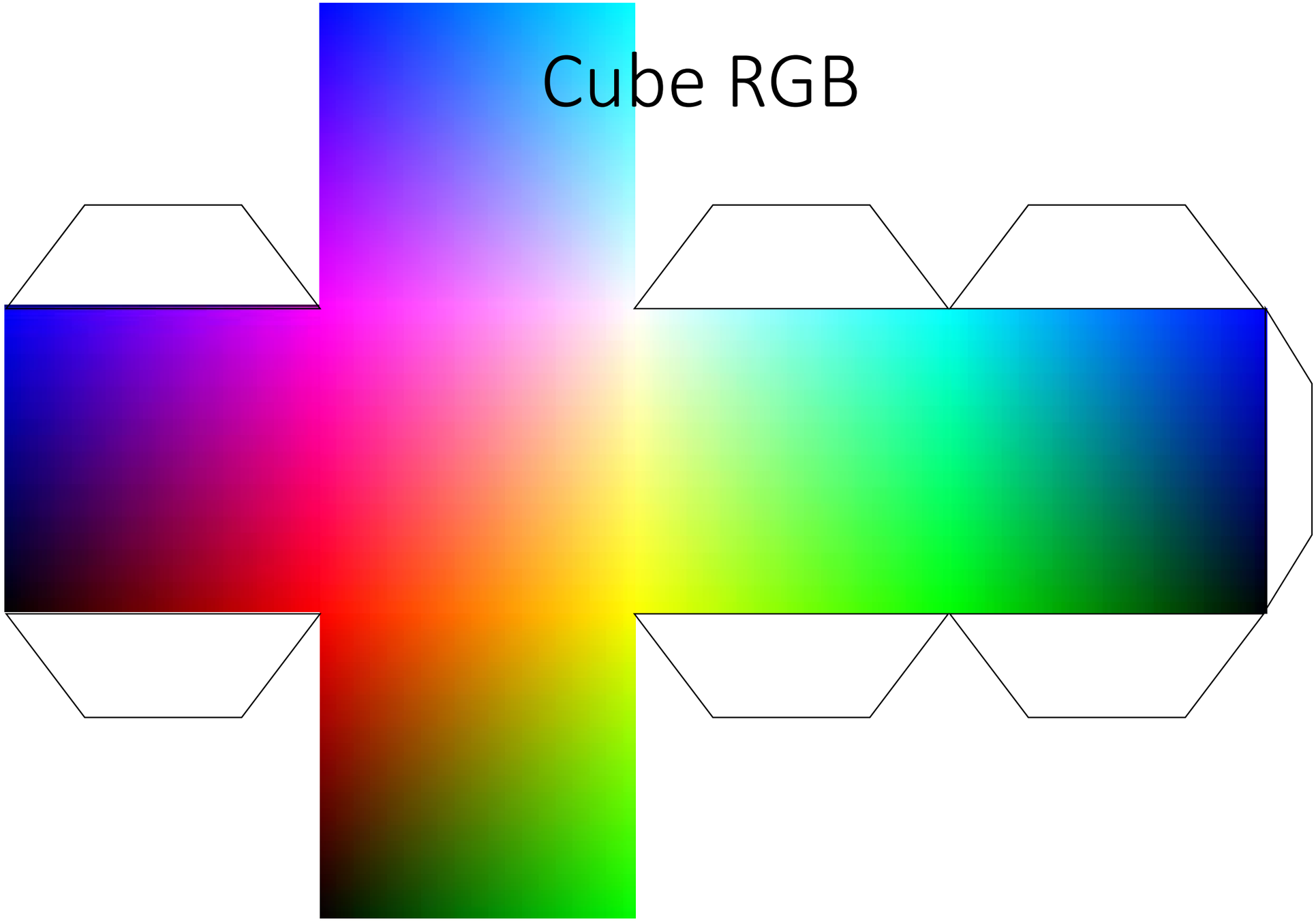
http://www.tu-harburg.de/rzt/tuinfo/periph/drucker/Color_Reproduktion/cie_spectrum.gif

Color spaces



http://www.gg.caltech.edu/~dhl/images.html

Cube RGB



Render methods

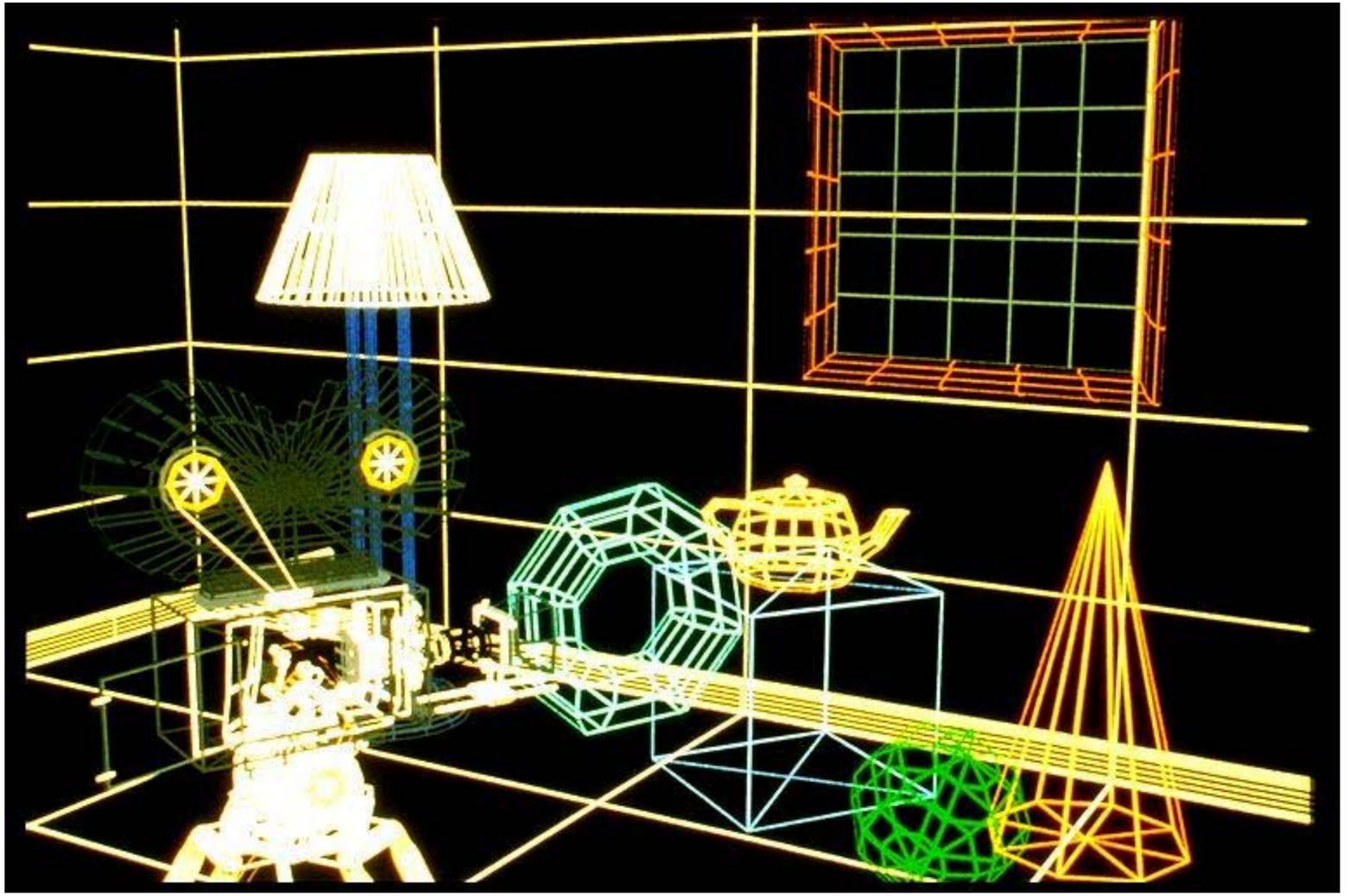
Contents

- Levels of rendering
- Wireframe
- Plain shadow
- Gouraud
- Phong
- Comparison Gouraud-Phong

Levels of rendering

- Wireframe
- Plain shadow
- Gouraud
- Phong
- Textures
- Global Illumination Models
  - Ray tracing
  - Radiosity

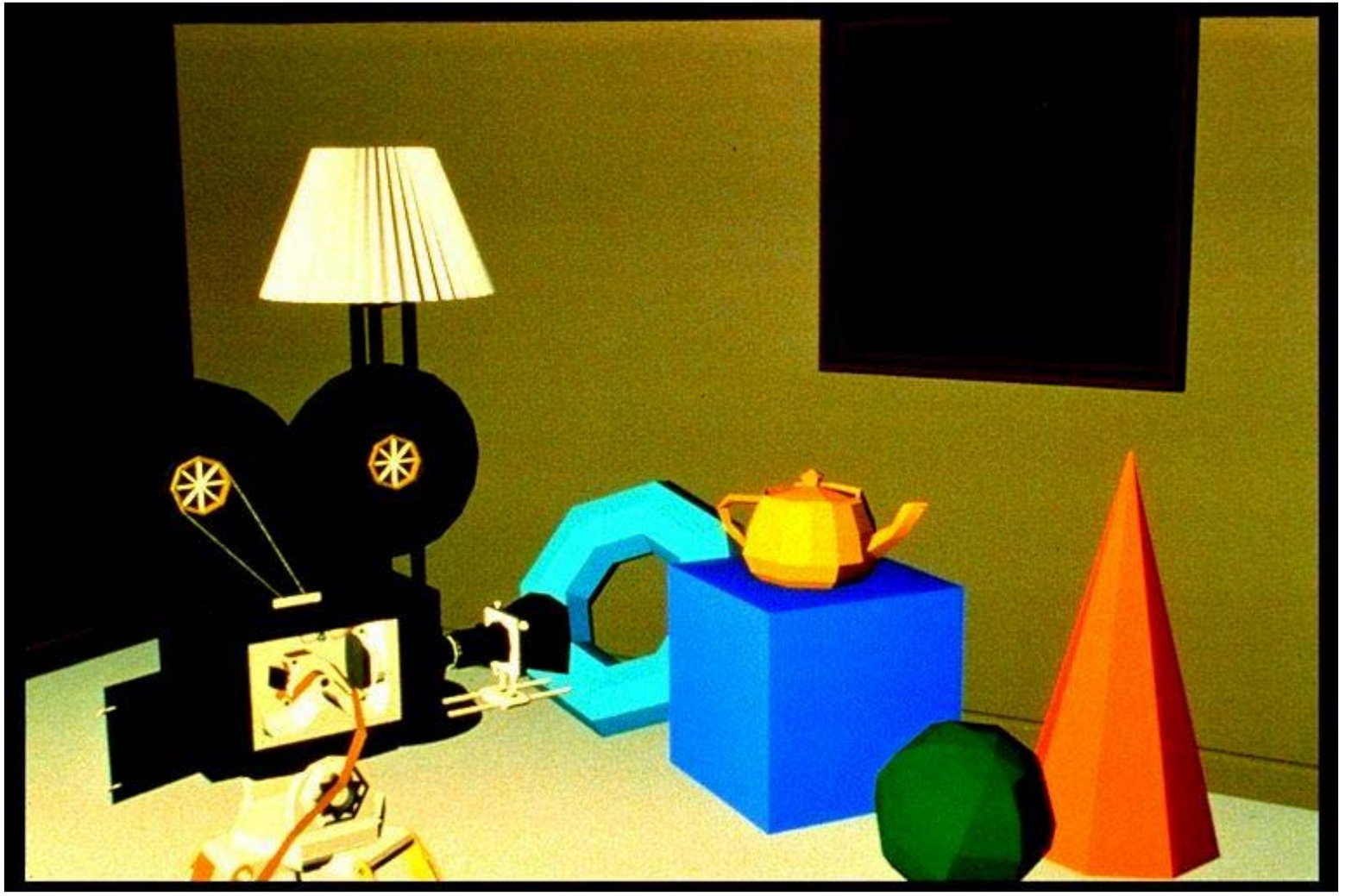
Wireframe - image



Plain shadow

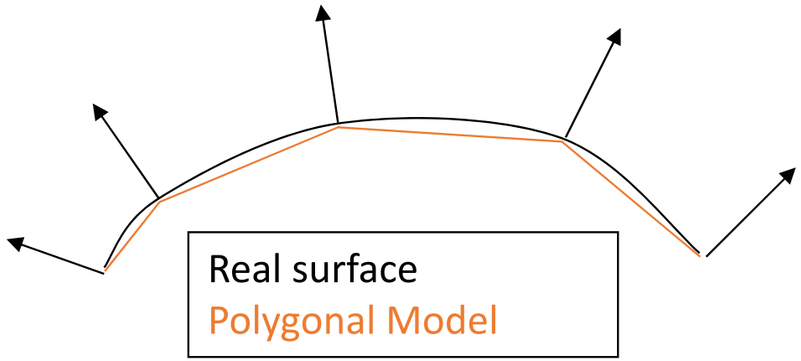
- Constant color for each polygon
- It applies the local illumination model to a point of the polygon (center)
- The normal vector is a date of the model or it can be calculated as the dot product of two edges
  - The direction (clockwise or counterclockwise) of the edges must be taken into account. And also the non-convex polygons.

Plain shadow - image



Gouraud

- It is based on the fact that the polygons make the approximation of a curved surface



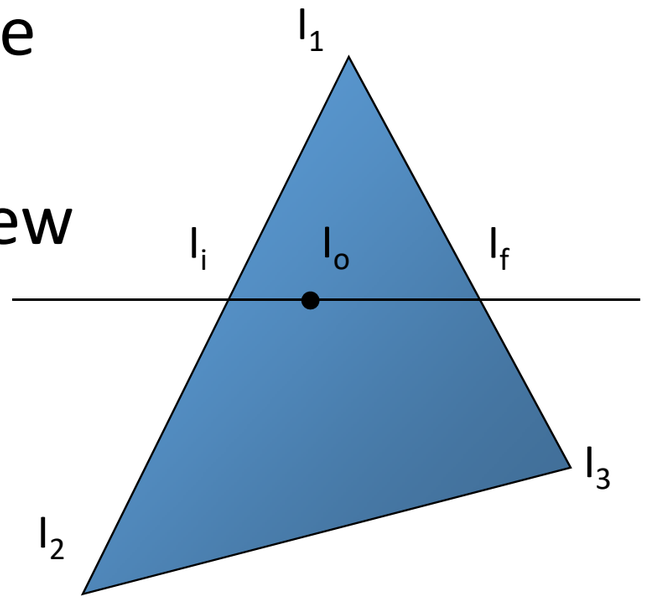
- It calculates the intensity in the vertex
- The intensities are interpolated between the vertex
- Each polygon is rendered

Gouraud – calculation of the normal vector

- The normal vector can be obtained:
  - From the original model
  - Calculated from the polygonal model
- From the polygonal model
  - The normal vectors of each polygon that shares the vertex are calculated:
    - Dot product of the edges
  - The average of the normal vectors is calculated

Gouraud - interpolation

- Interpolation of the intensities calculated in the vertex
- The intensities are calculated in the global system
- The interpolation is done in the view system
  - Interpolation between edges
  - Interpolation in the scan line



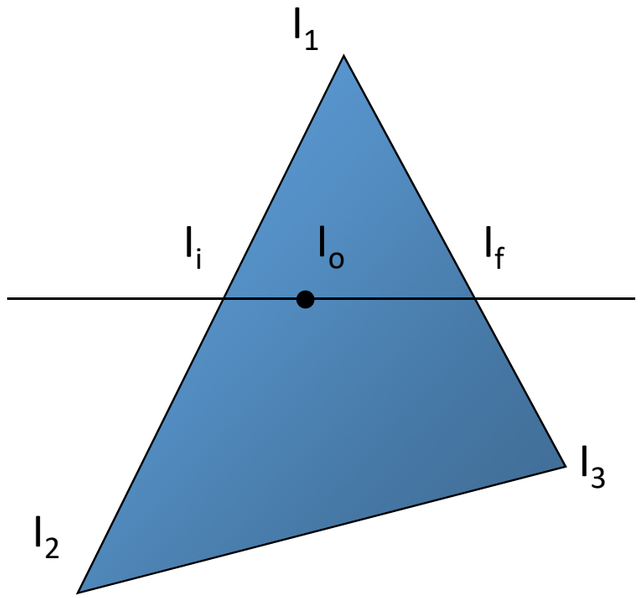
Gouraud - formulas

$$I_{1,2,3} = k_a I_a + k_d I_p (n \bullet l) + k_s I_p (r \bullet v)^s$$

$$I_i = I_1 \frac{y_o - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_o}{y_1 - y_2}$$

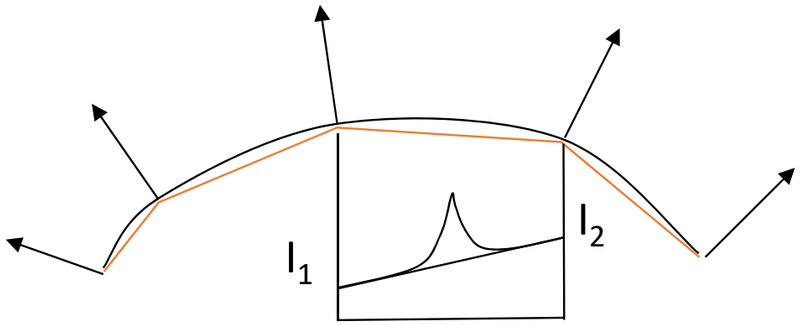
$$I_f = I_1 \frac{y_o - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_o}{y_1 - y_3}$$

$$I_o = I_i \frac{x_f - x_o}{x_f - x_i} + I_f \frac{x_o - x_i}{x_f - x_i}$$



Gouraud - problems

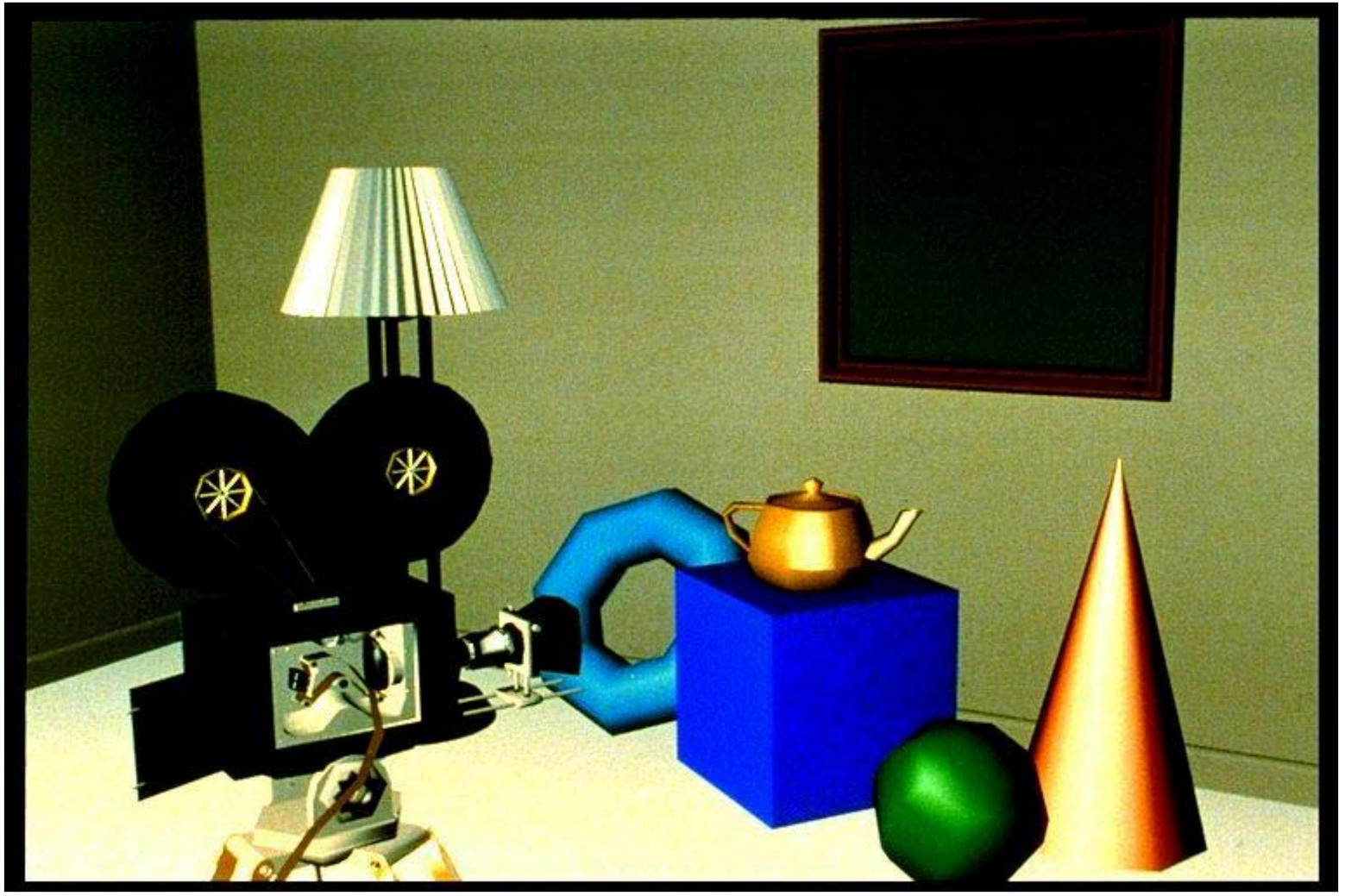
- It doesn’t render the specular light
  - In the interpolation process, the maximums are in the vertex
  - In the real model the specular light only affects a small area, so it is difficult that it appears in the interpolation



Gouraud - problems (2)

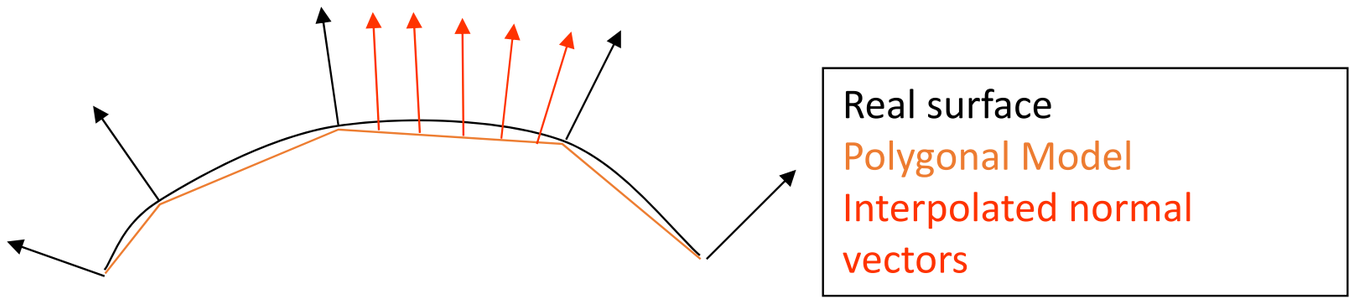
- It is possible that the local model is not coherent
  - It can emit more light than it receives
- In view coordinates, the “x” and “y” lengths are not linear in relation to “z”
- Polygonal shapes
  - It can be solved with more detailed meshes.
  - It can be solved using parametric surfaces

Gouraud - image



Phong

- It interpolates the normal vectors instead of the intensities
- It represents a polygonal model with a normal vector in each point
- It solves the specular light problem



Phong - problems

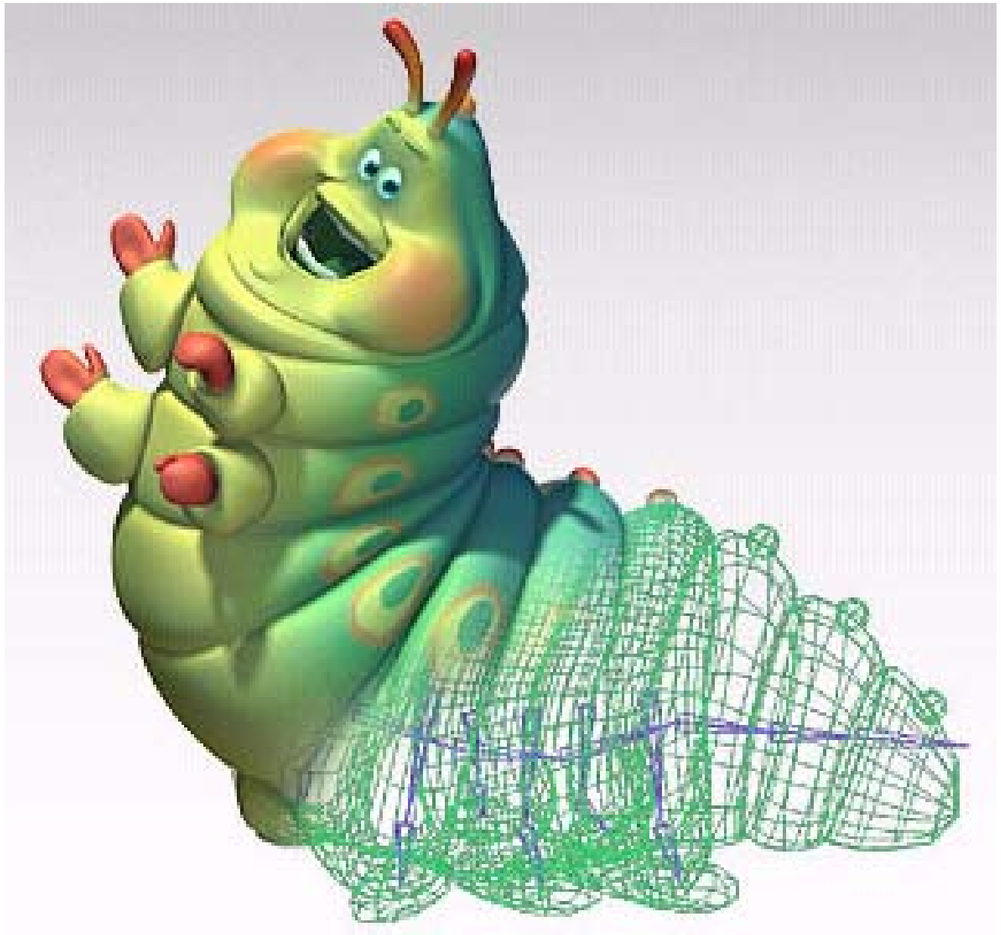
- In terms of computation, it is more expensive
  - It interpolates vectors instead of scalar values
  - It calculates the intensity in each point
- The other problems remain

Phong - image

Gouraud-Phong comparison

- Gouraud is less expensive but it doesn’t render correctly the specular light (it is different in function of the orientation)
- Phong solves the problem with more calculations
- A Gouraud shading can be used for objects with diffuse reflection and a Phong shading for objects with specular reflection

Rendering



Contents

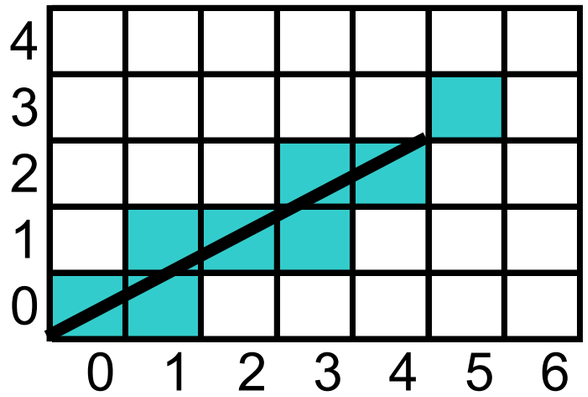
- Objective
- Rendering of lines
  - DDA algorithm
  - Bresenham algorithm
- Polygons rendering
- Polygons filling
- Visible surface detection

Rendering

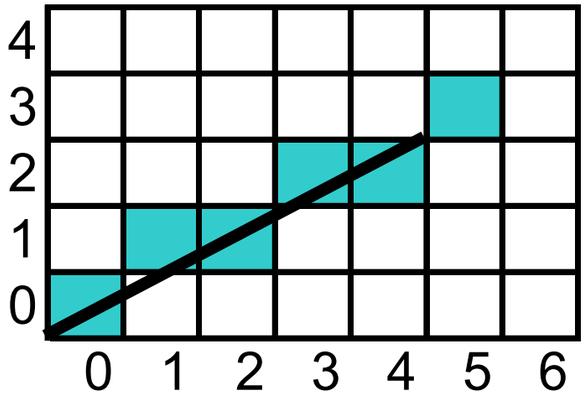
- There is information about:
  - Topology of the scene
  - Coordinates of the projected vertexes
  - Intensity in the vertex or in each point
  - Z coordinate in the vertex
- The objective is to calculate:
  - Color in each pixel

Rendering of lines

- Input data: the coordinates of the two vertex
- It must be calculated the pixels that must be set
  - It produces an incorrect solution to set all the pixels where the line goes through.



Incorrect solution



Correct solution

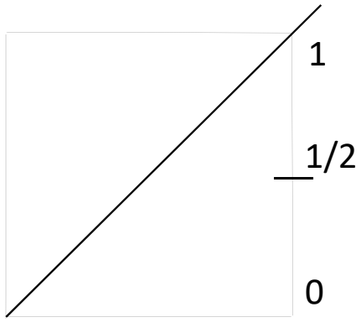
DDA algorith (Digital Diferential Analyzer)

```
dx = xb - xa
dy = yb - ya

x = xa
y = ya

If (Abs(dx) > Abs(dy)) Then
  steps = Abs(dx)
Else
  steps = Abs(dy)
End If
```

```
xIncrement = dx / steps
yIncrement = dy / steps

Call Plot(x, y)

For k = 0 To steps - 1
    x = x + xIncrement
    y = y + yIncrement
    Call Plot(x, y)
Next k
```

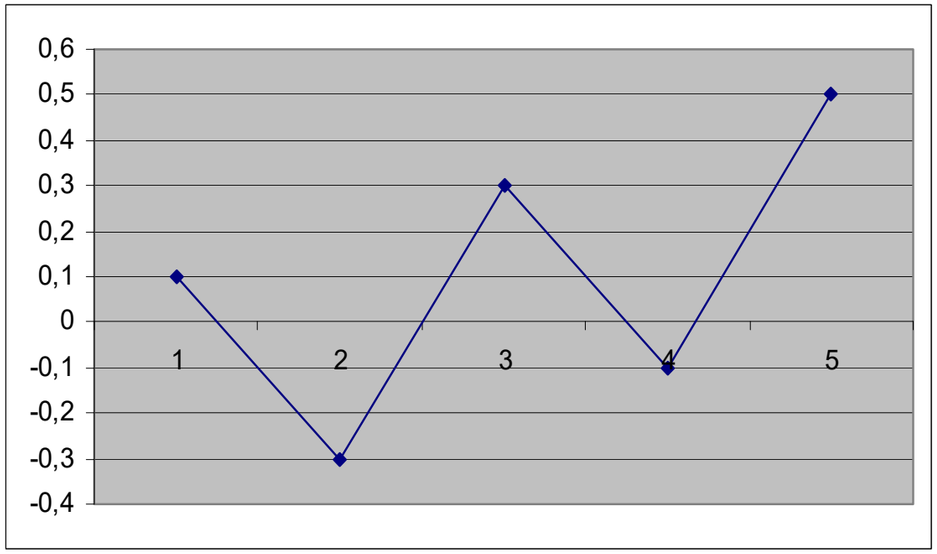
Bresenham algorithm

- DDA works with real numbers
- Bresenham algorithm was developed for digital plotters
- It is based on:
  - Incrementing the biggest dimension
  - The other coordinate increments 0 or 1
  - The error is controlled with the difference between the line and the start of the nearest pixel



Bresenham algorithm

- The error increments with the value of the slope: e = e + dy/dx
- When error > 1/2:
  - It increments y
  - It subtracts 1 from error
- The process starts with: e = -1/2
  - The control is made with e > 0



Bresenham algorithm

- Operations with e:
  - initialization: e = dy/dx - 1/2
  - increments: e = e + dy/dx
  - Control: if (e>0) then e = e -1, x = x + 1
- x and y are integer
- e is real
  - To work with integers,
    we multiply error by 2 • dx

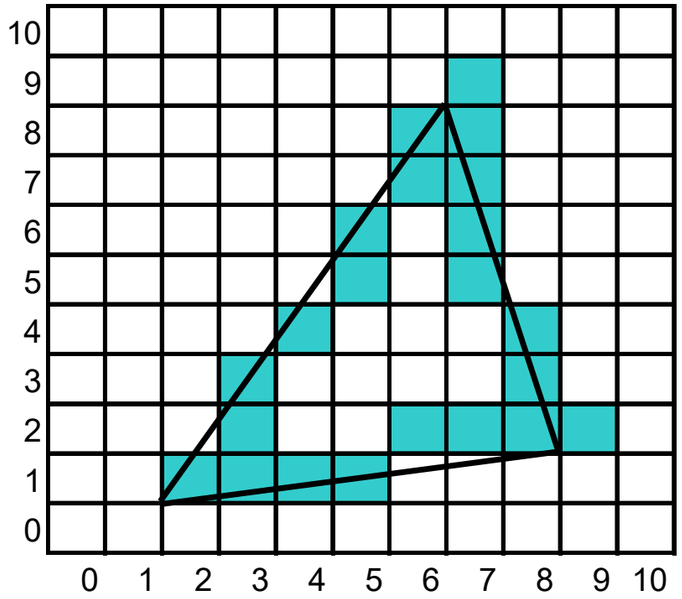
Bresenham algorithm

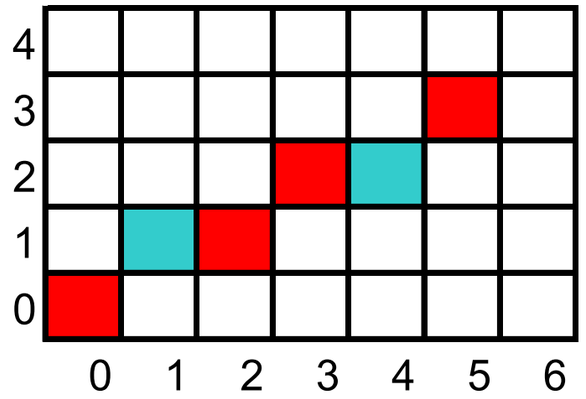
```
dx = Abs(xa - xb)
dy = Abs(ya - yb)

e = 2 * dy - dx

If (xa > xb) Then
  x = xb
  y = yb
  xEnd = xa
Else
  x = xa
  y = ya
  xEnd = xb
End If
```

```
Call Plot(x, y)

Do While (x < xEnd)
   x = x + 1
   If (e > 0) Then
     y = y + 1
     e = e - 2 * dx
   End If
   e = e + 2 * dy
  Call Plot(x, y)
Loop
```

Polygons rendering

- It is processed line by line (scan line)
- It is filled inside the start and end of each edge



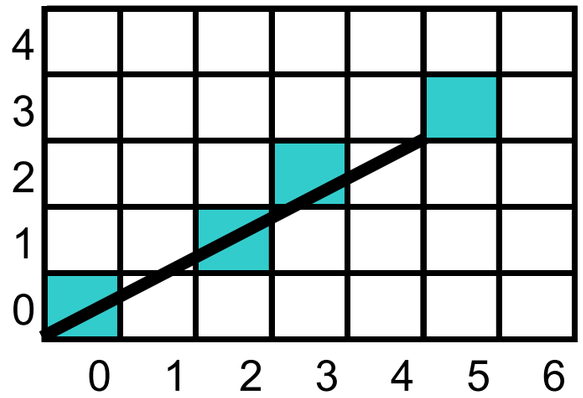


For the polygon, only the red pixels are need

Rendering of edges of polygons

- There is need a pixel by horizontal line (scan line)
- The algorithms are modifications of the DDA or Bresenham algorithms, example with modification of DDA:

```
dx = xb - xa
dy = yb - ya

x = xa
increment = dx / dy

For y = ya To yb
    Call Plot(x, y)
    x = x + increment
Next y
```



Filling between edges

- In each line (scan line) there is an even number of edges.
- In convex polygons there are always 2 edges.
- In the filling process, the shading and z coordinate is calculated.
- The rendering can be done in two ways:
  - scan line (one line at a time)
  - polygon by polygon

Scan line

- The image is calculated line by line, starting usually by the top line.
- In each line:
  - The process calculates the list of edges in that line (it is done adding new edges and eliminating the edges that have ended)
  - In each pixel:
    - Calculate the values of the polygon in this pixel, interpolating from the values in the edges (incrementally)
    - Render the values of the pixel nearest to the point of view

Polygon by polygon

- The image is generated polygon by polygon.
- In each polygon:
  - For each horizontal line between y_{max} and y_{min} of the polygon:
    - Get the list of edges in the line.
    - Render the pixels between this edges if this is the nearest polygon to the point of view (the z value is stored)

Elimination of hidden surfaces

- Historically, there has been different methods to solve this problem.
- One of the most known methods it the painter’s algorithm: paint the polygons from furthest to closest.
- Currently, the Z-buffer is used
  - It is the method can be used in the rendering polygon by polygon
  - It is implemented in the hardware of graphic cards.

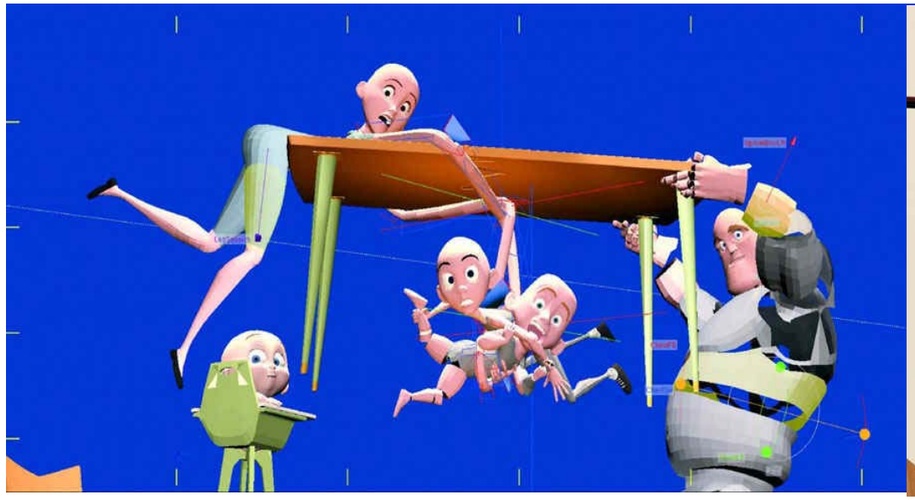
Z buffer

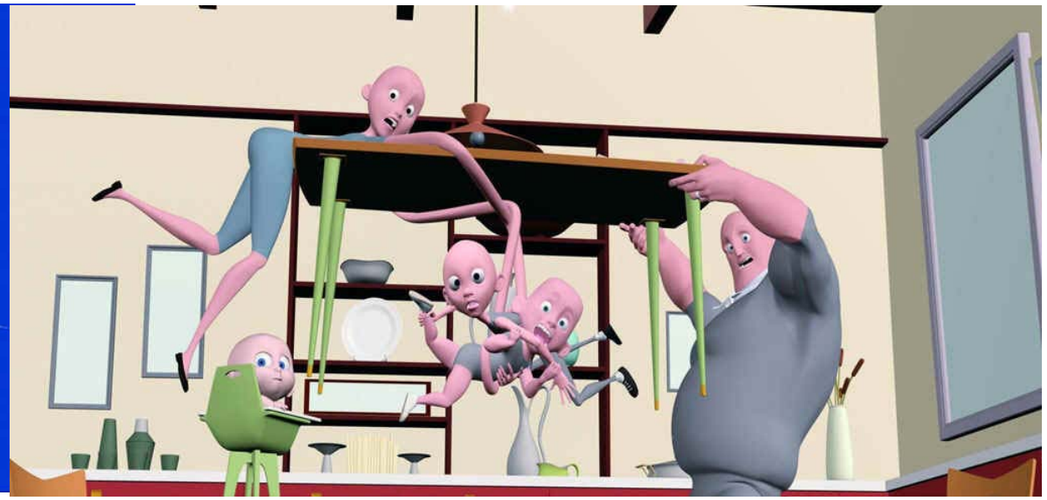
- It uses a matrix with the values of the z coordinate for each pixel
- It allows rendering the polygons one on one.
- When rendering a polygon, the value of the z coordinate is compared with the z value stored for that pixel, if it is closest, it is rendered and their z value stored in the matrix.

Scenes creation

- Storyboard
- Models of the objects
- Position
- Initial rendering
- Modifications
- Rendering final

http://www.pixar.com/howwedoit/index.ht





Modeling of objects

http://www.theforce.net/scifi3d/starwars/meshes/milfalcon.jpg

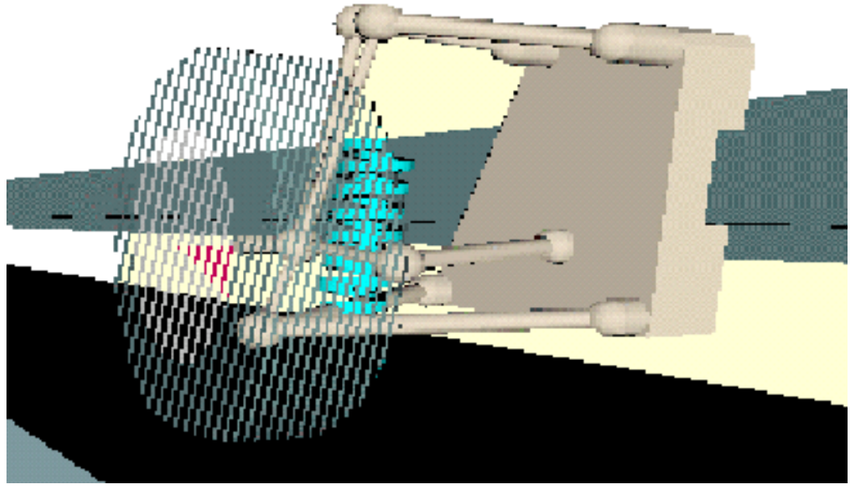
- Watt chap. 2, Hearn 10.1 - 10.4, 10.14-10.16

Contents

- Objects building
- Rendering of objects
- Polygonal representation
  - structure, creation, mesh, attributes, ...
- Others methods (splines, CGS, volumetric).

Goals

- Rendering of a real object or existing as a model in the computer.
- To render the image or to study other actions (CAM, analysis)

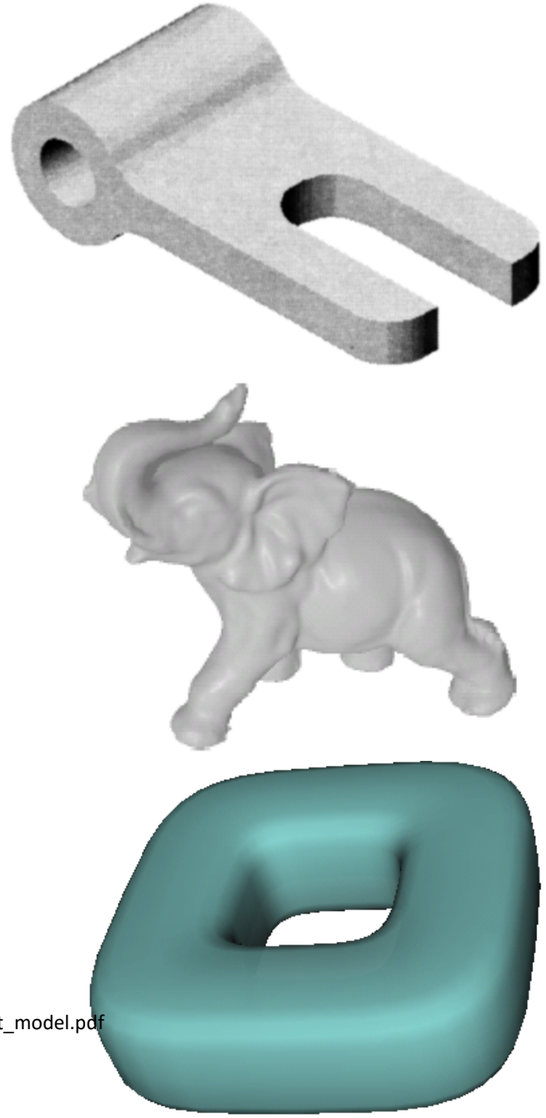


Functions

- The modelization includes:
  - Data structure to render the object
  - Creation of the object in the computer
  - Edition of the object

Creation of objects

- With a CAD interface (3DStudio, ProEngineer,…)
- From real objects (laser explorer, 3D digitizer)
- Matematically

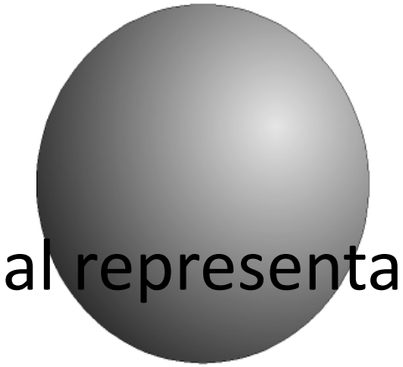


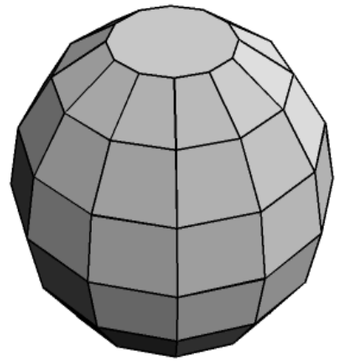
http://graphics.stanford.edu/papers/rt_model/rt_model.pdf

Visualization Toolkit

Object representation

- There isn’t a unique method
- It depends on the object, purpose and media:
  - User interface
  - Computer representation
  - storage
- The most popular: polygonal representation (computer representation)

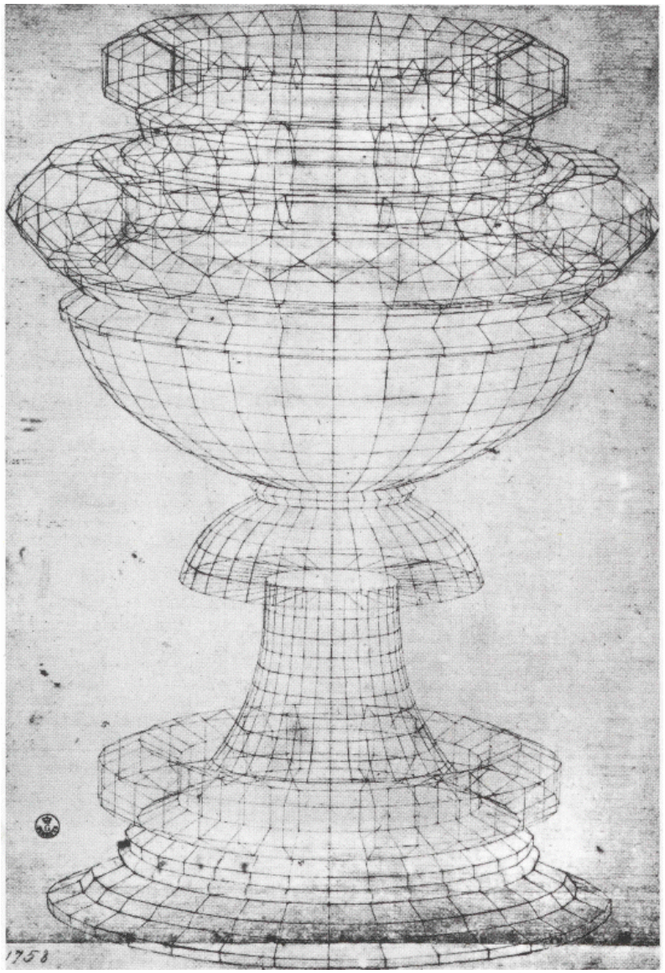




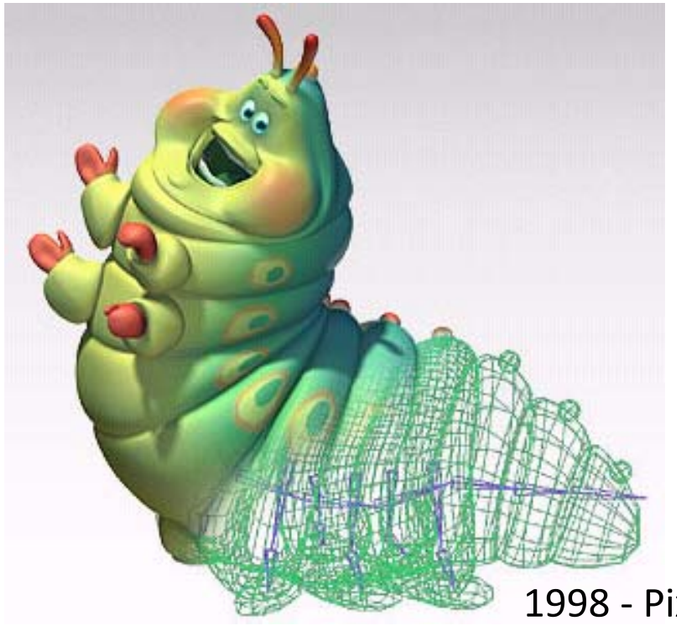
Modeling Methods

- Polygonal representation
- Splines
- CSG (Constructive Solid Geometry)
- Space partitioning (octrees y BSP: binary space partitioning)
- Implicit representation

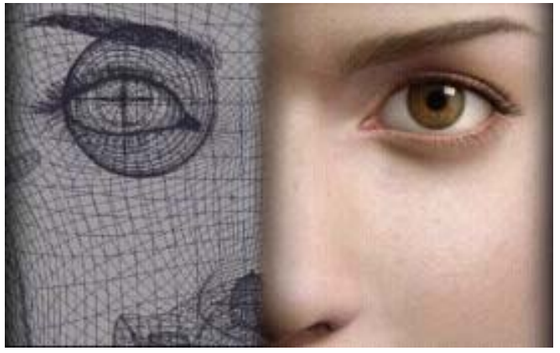
Polygonal representation



1430 - Florencia



1998 - Pixar (A bug’s life)



2001 - Final Fantasy

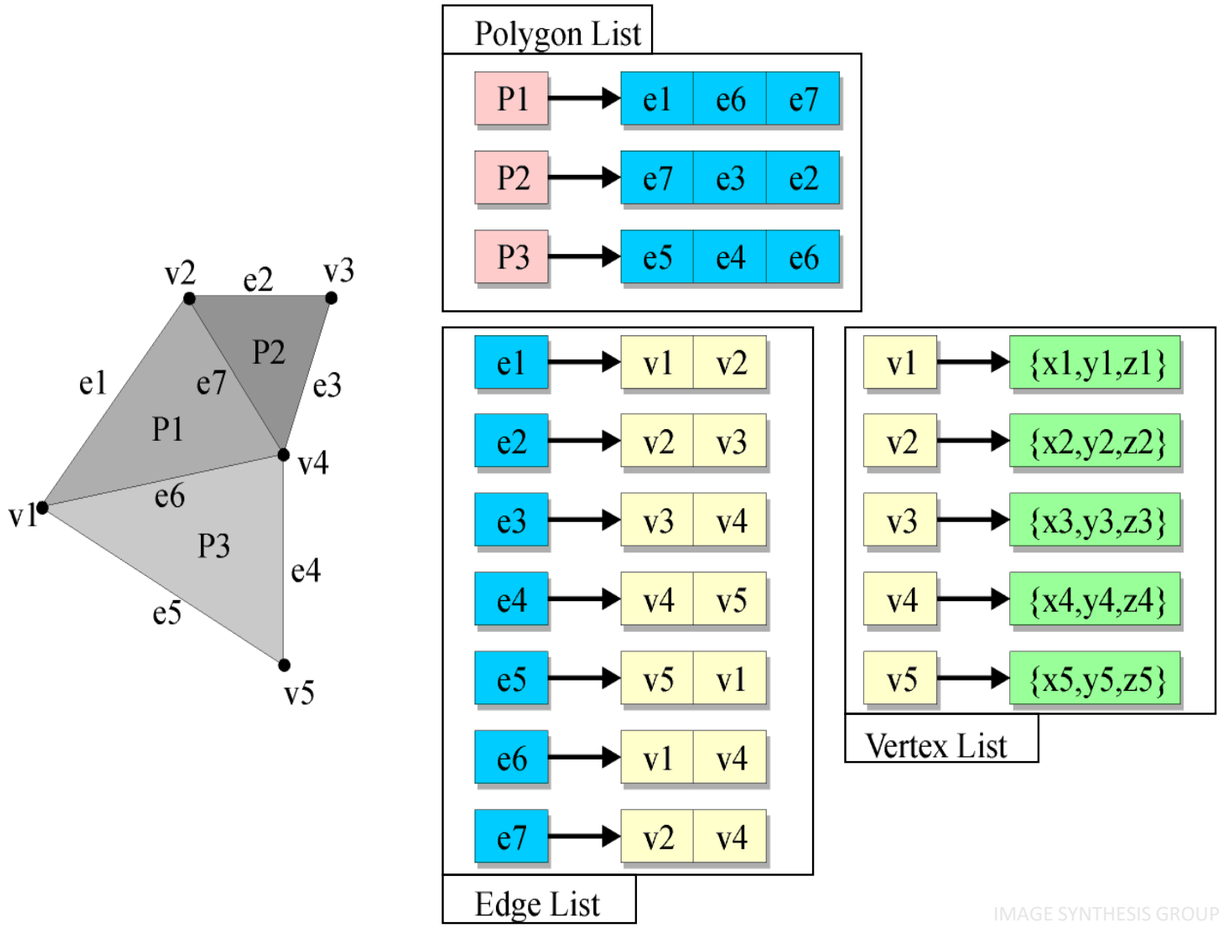
Polygonal - Features

- It can render any object
- It is the method used by the graphic hardware (graphic cards)
- It is difficult to edit after creation.
- It requires a lot of information
- It is used together textures

Polygonal - Features

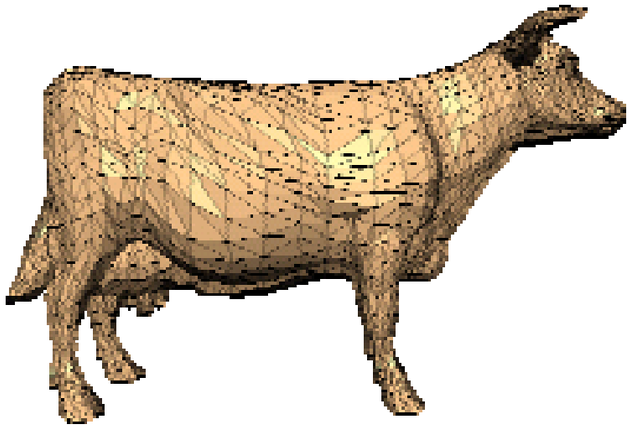
- It is necessary to differentiate surfaces and polygons
  - Example: a closed cylinder: 3 surfaces, n polygons
- In curved surfaces, it is an approximation
- The number of polygons depends of the curvature
- Usually triangles are used.
- Equation of the plane: $Ax + By + Cz + D = 0$

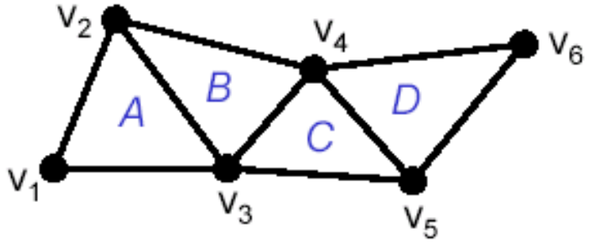
Polygonal – Data structure

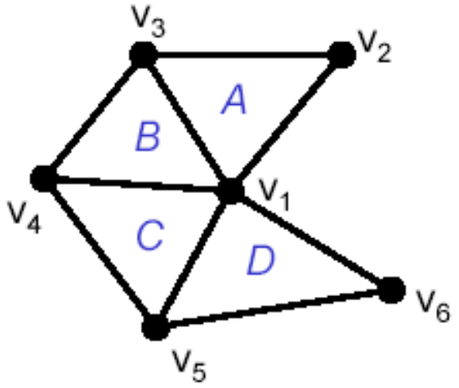


Polygonal - Meshes

- The vertexes are shared by several polygons
- Optimization using meshes
  - strips (triangle - strips)
  - fans (triangle - fans)







Poligonal - Atributos

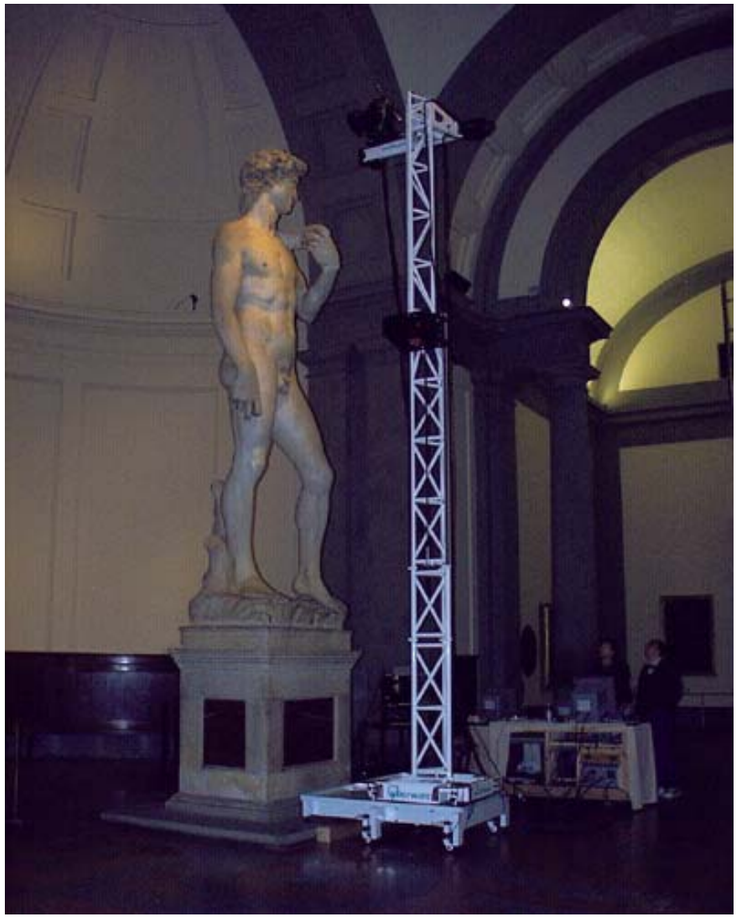
- Atributos de polígonos
  - triangular o no
  - area
  - normal
  - ecuation del plano
  - convexo o no
  - con agujeros o sin ellos
- Atributos de las aristas
  - longitud
  - arista de superficie o polígono
- Atributos de los vértices
  - lista de polígonos
  - valor del sombreado
  - normal
  - coordenadas de textura

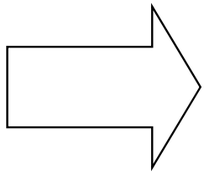
Poligonal - Generation

- Modelado
- Exploration de objects reales
- Generation matemática
  - Funciones
  - Extrusión y perfiles
- A partir de otros modelos de represe
  (CSG, splines, voxels)

http://www.sun.com/960710/feature3/alice.html

Poligonales - Exploration

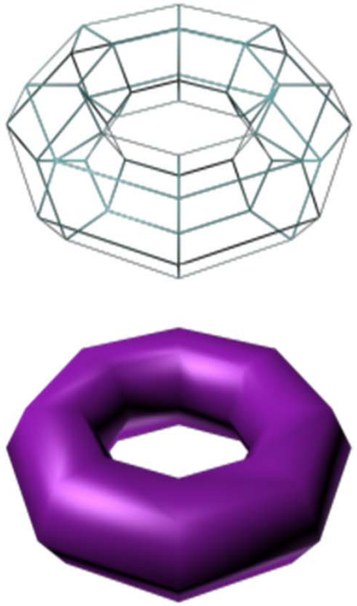




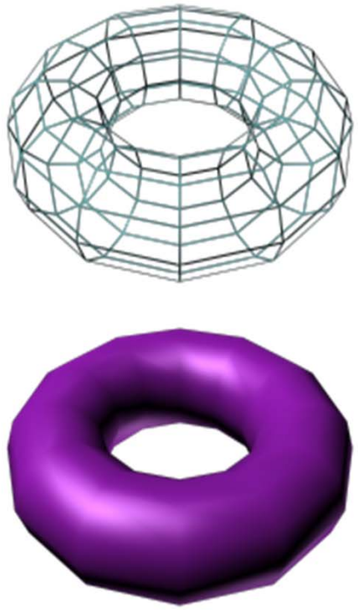
http://graphics.stanford.edu/projects/mich/mich.html

Poligonal - Resolution

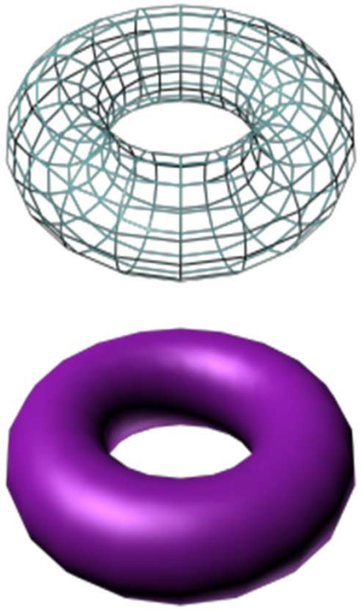
48 polígonos



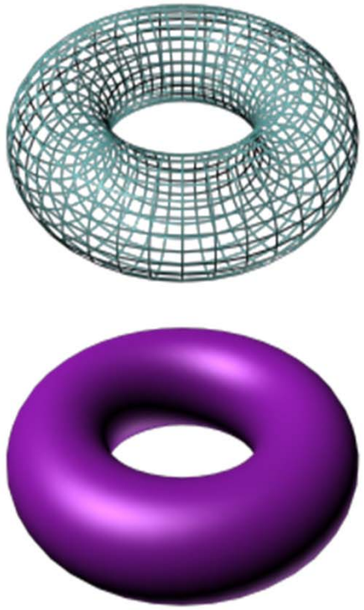
120 polígonos



300 polígonos



1000 polígonos

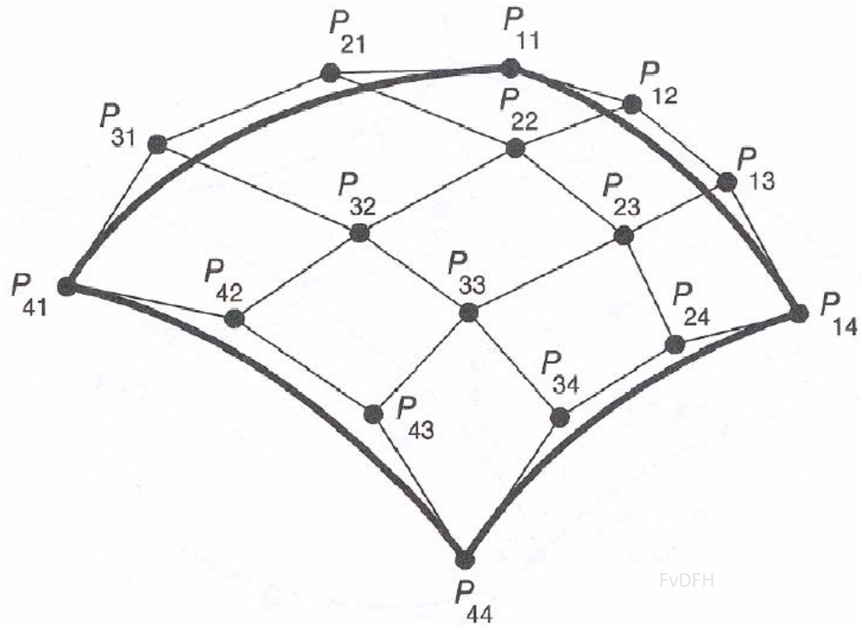


Poligonal - Fractales

- Misma forma en infinitas resoluciones
- Basada en subdivisión de polígonos

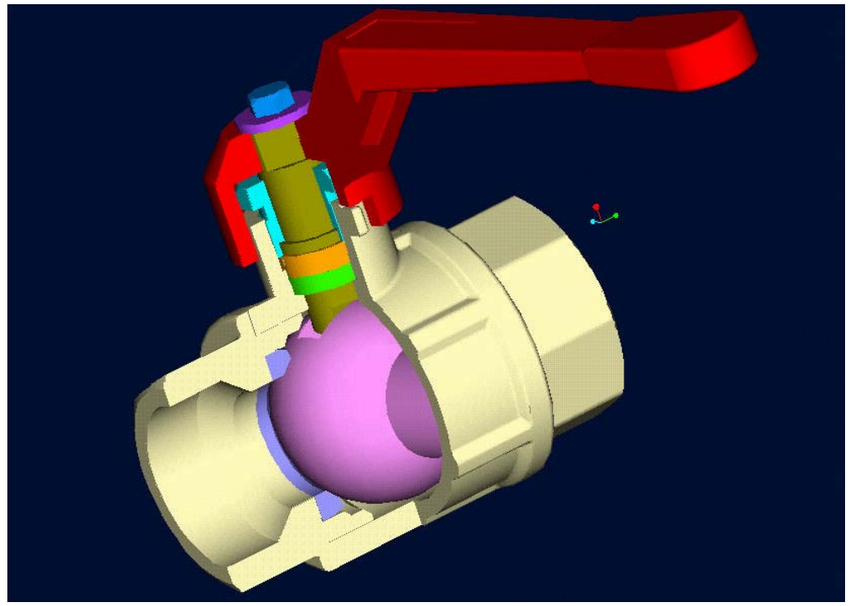
Splines

- Spline: banda flexible que produce una curva suave a través de un conjunto de puntos
- Curva con secciones polinómicas
- Diseñar curvas y super-ficies

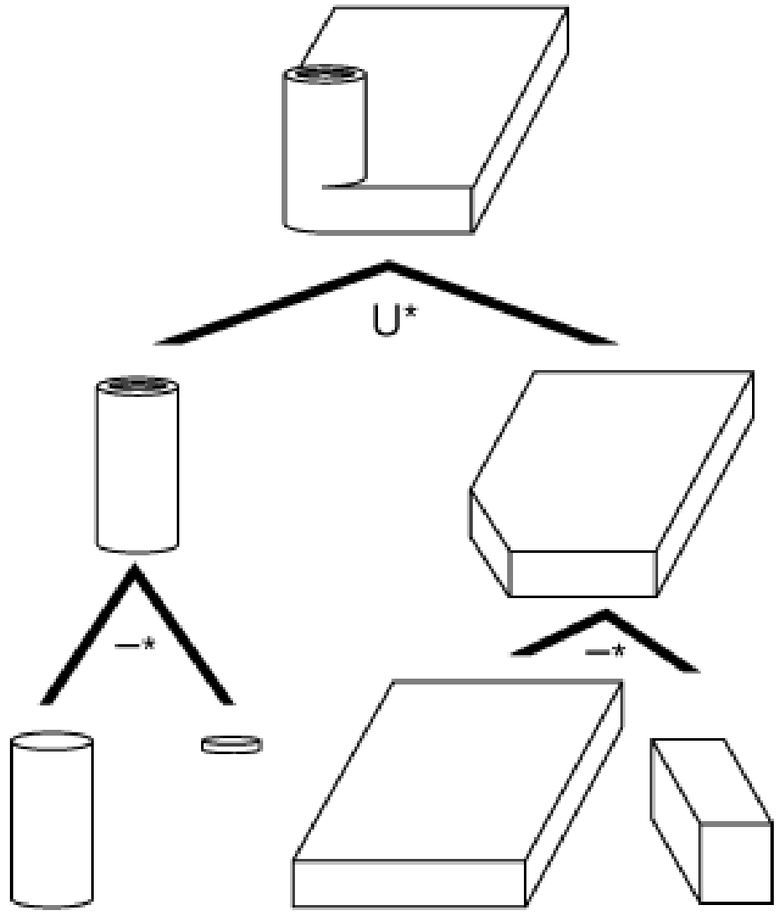


Constructive Solid Geometry

- Primitivas, transformaciones y operaciones booleanas
- Primitivas: cubo, esfera, pirámide, cono, ...
- Operaciones booleanas
  - Unión
  - Substraction
  - Intersection

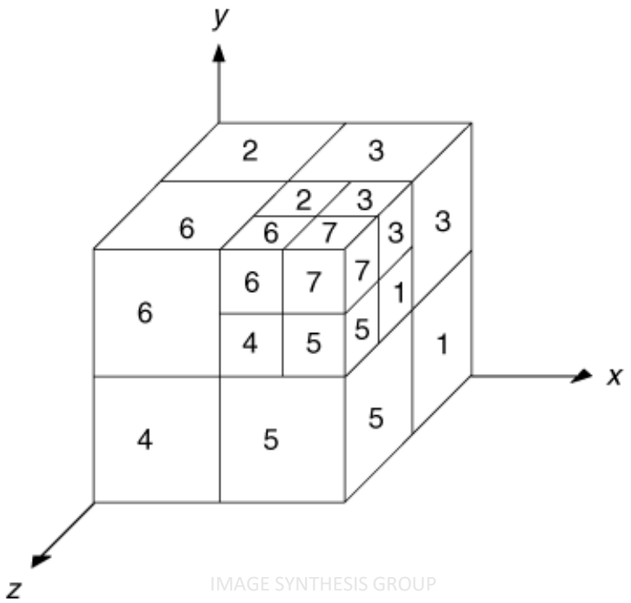


CSG - Ejemplo



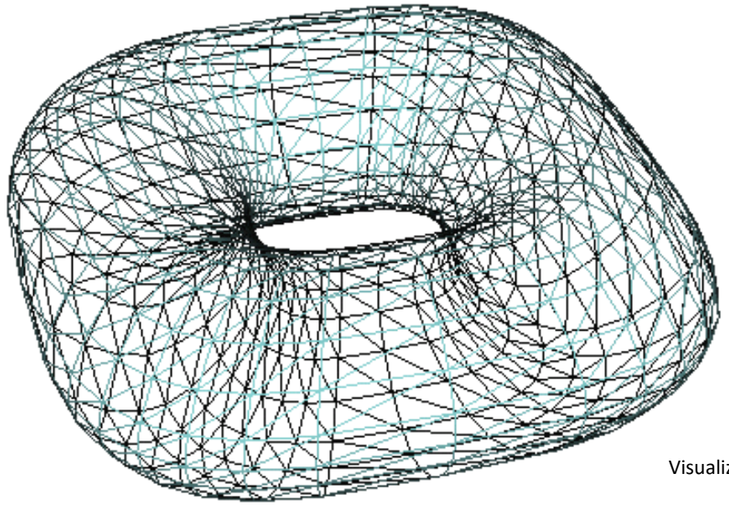
Subdivisión del espacio

- Son métodos volumétricos
  - Voxels: bitmaps en 3 dimensiones
  - Octrees: división en 8 cubos
  - BSP: división binaria del espacio



Funciones implícitas

- Restringido a ciertos objects
  - ejemplo: primitivas de CSG
- Cuadrícas (esfera, elipsoide, toro, ...)
- Supercuadrícas (incorporan parámetros adicionales)



Visualization Toolkit

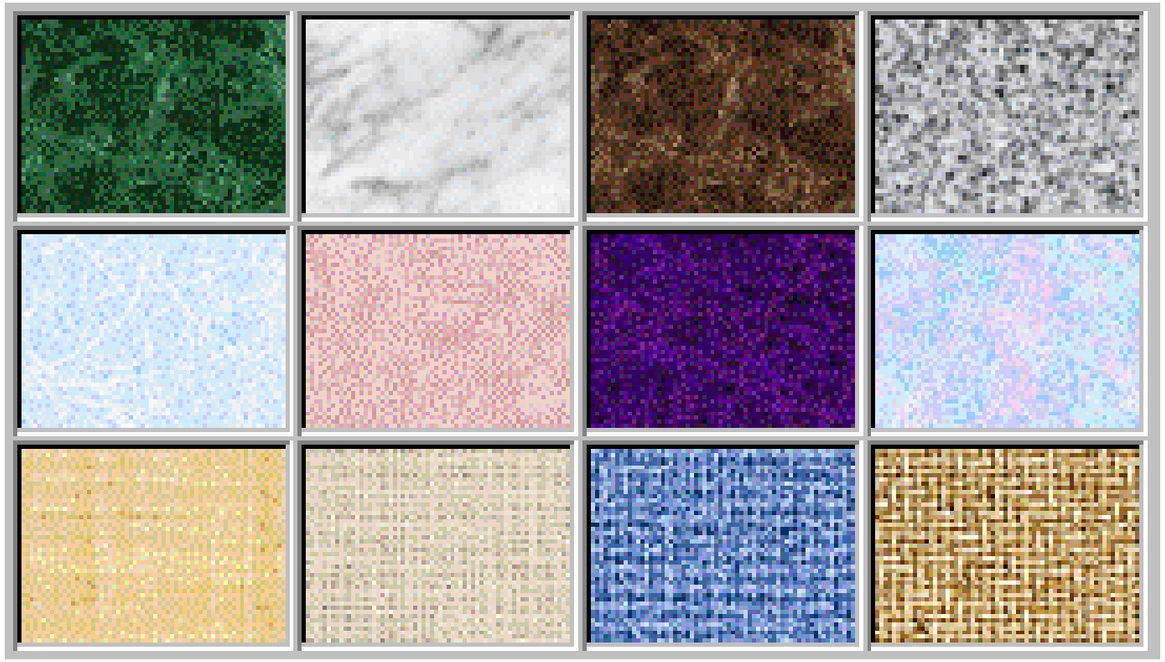
Gestión de la escena

- Se representan escenas, no solo objects
- Estructuras jerárquicas
- Aplicaciones de tiempo real
- objects representados por hardware y el software indica qué objects
- Nivel de detalle de los objects

Otras clasificaciones

- Representation de bordes (Boundary Rerepresentation: B-reps)
  - Define un object por las superficies que separan el interior del object del entorno
- Partition del espacio
  - Conjunto de sólidos continuos no solapados

Texturas



Contenido

- Concepto de textura
- Utilización
  - Mapeado de color
  - Mapeado del entrono
  - Bump mapping

Introducción

- Mapeado de texturas: mapear una imagen bidimensional en un objeto
- El sombreado de Phong produce objetos de apariencia plástica
- Los métodos para dotarle de realismo son:
  - texturas (añadido a los algoritmos tradicionales)
  - iluminación global (nuevos algoritmos)

Utilización de texturas

- Mapeado de color
- Mapeado del entorno
- Bump mapping
- Transparencias

Fundamentos

- Proceso:
  - Se asocia la textura a la superficie del objeto
  - Se proyecta el objeto en la ventana
- Es una transformación de 2D a 2D
- Se realiza en dos fases
  - parametrización
  - proyección

Parametrización

- En objetos poligonales se asocian a los vértices las coordenadas de textura (u, v)
- La imágen de la textura tiene unas coordenadas u, v

$P_i(x, y, z, u, v)$

Mapeado inverso

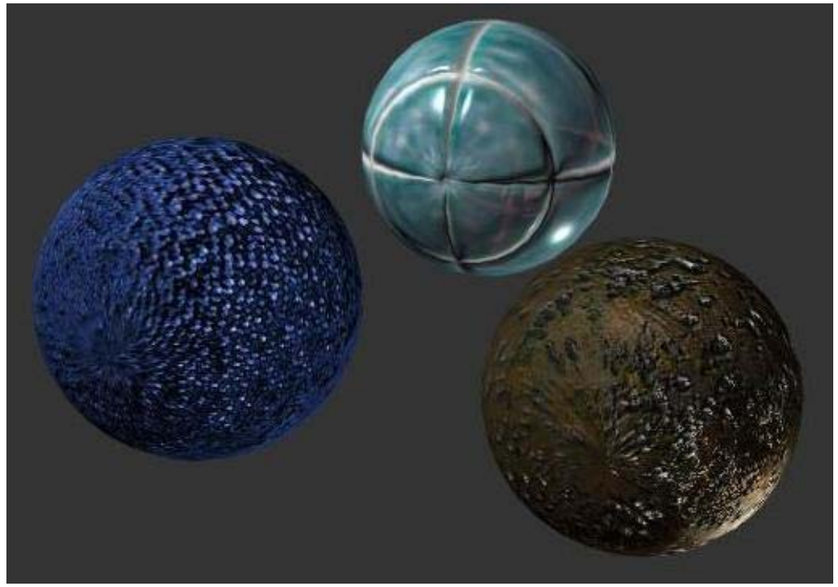
- En el proceso de rendering se recorre cada pixel de la ventana de salida
- Se calcula el valor de u y v
  - matriz de transformación
  - más habitual, interpolación bilinear
- Necesidad de anti-aliasing, debido a que un pixel de la ventana de salidad puede corresponder a un varios pixels de la textura

Texturas- imagen

Bump mapping

- Desarrollada en 1978 por Blinn
- Muestra deformaciones sin necesidad de modelarlas

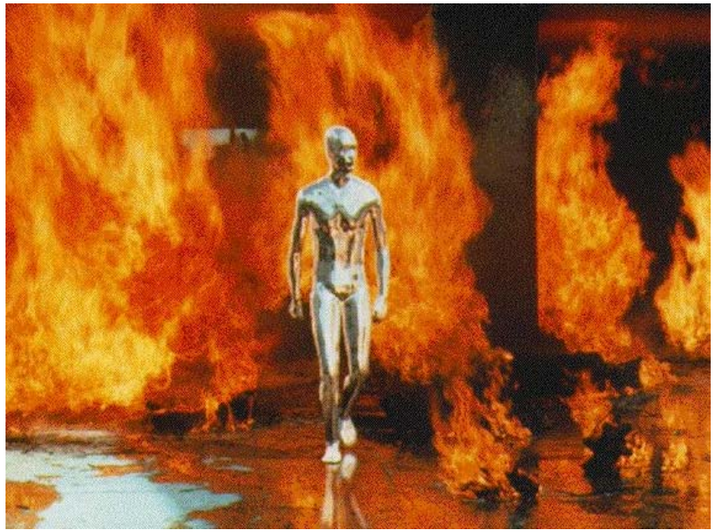
- Deforma las normales de la superficie
- Se aprecia en el contorno del objeto
  - sigue siendo el original



Bump mapping - imagen

Mapeado del entorno

- Environment mapping, reflection mapping, chrome mapping
- Consiste en reflejar el entorno del objeto
- La textura se mueve con el objeto



http://www.debevec.org/ReflectionMapping/

Ejemplos de mapeado del entorno

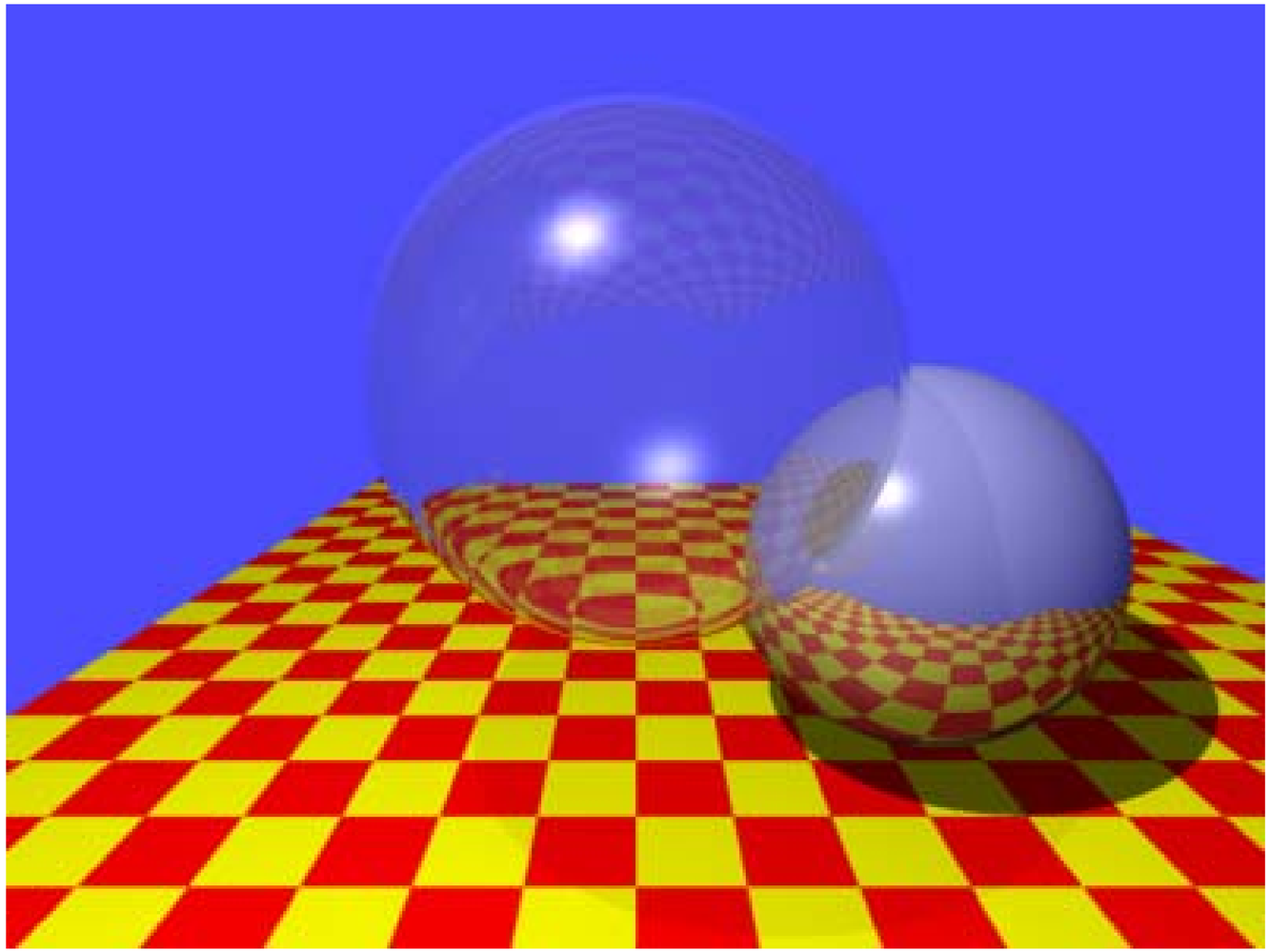
Flight of the Navigator - 1986

Star Wars Episode I: The Phantom Menace

http://www.angelfire.com/scifi/spacecraft/fnspacecraft/nabooroyal.htm

Mapeado del entorno - imagen

Global Illumination Models



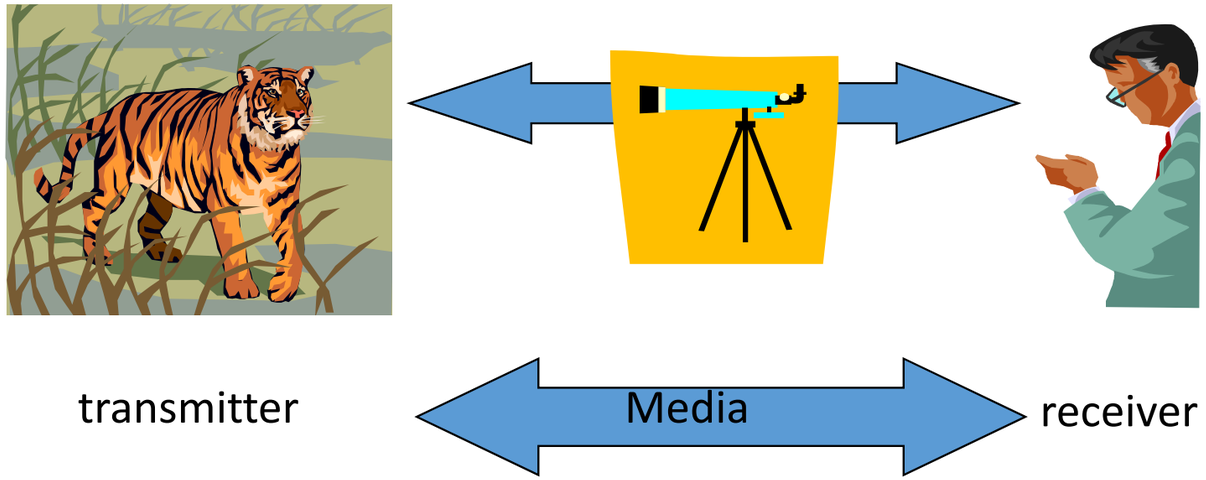
THE WHITTED IMAGE - BASIC RECURSIVE RAY TRACING Copyright © 1997 A. Watt and L. Cooper

Content

- Reality and perception
- Local and global illumination
- The rendering equation
- Algorithms
  - Ray tracing
  - Radiosity
- Applications
  - POV-Ray, Radiance

Reality and perception

- We know the reality because it sends us messages about how it is
- Communication



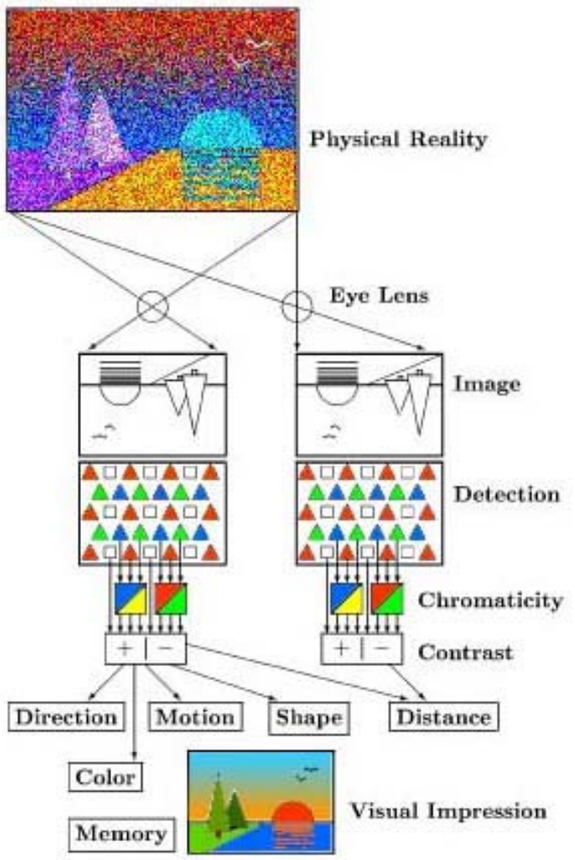
The message is the information of the own transmitter

Knowledge of the reality

- We know the reality with sensors (receptors)
  - Sight, hearing, sense of touch, taste, smell
- With the sight sense, we interpret 4 features of the objects:
  - Form
  - Position
  - Illumination (Bright and color)
  - Movement

Visual Path

- The light arrives to the eye from the object
- It is projected in the retina
- The receptors
- The receptors detect color and light
- They interpret
  - Illumination (color and contrast)
  - The contrast define the shapes
  - The situation of the receptors and the comparison of both eyes define the position
  - The sequence of images gives information of the movement



Perception

- Mental model of the real world
- It is built from the stimulus of the senses and interpreted by our brain
- The principal feature is the recognition of patterns

Global Illumination

- It considers the reflected light by a point taking into account all the light that arrives
- Not only the incident from the source lights
- Effects
  - shadows
  - reflection of an object in others
  - transparencies

Reality and illumination

- The illumination depends on the transmitter and receiver

Models of global illumination

- Ray tracing
  - Specular interactions
- Radiosity
  - Diffuse interactions
- Most of the algorithms use both

The rendering equation

- Equation introduced by Kajiya (1986)

$$I(x, x') = g(x, x')[\varepsilon(x, x') + \int_s \rho(x, x', x'') I(x', x'')dx'']$$

- $I(x, x')$: total amount of light from x' to x
- $g(x, x')$: function of visibility, 0 or proportional to the inverse of the square of the distance
- $\varepsilon(x, x')$: emitted light from x' to x
- $\rho(x, x', x'')$: bidirectional reflectance distribution function, light reflected from x'' to x in x'

The rendering equation (2)

- We need the functions:
  - Visibility
  - Emission
  - Bidirectional reflectance distribution
- The integral is not analytic
- It is independent of the point of view
- It is for all the points, not only the incident rays on the eye
- It is recursive

Global illumination algorithms

- Basic solution:
  - From a light source, we emit all the possible light rays and follow their path until they arrive to the point of view or go out of the scene
- Approximations:
  - Use only specular and diffuse interactions
  - Take into account only a subset of the light rays emitted by the source lights

Ray tracing

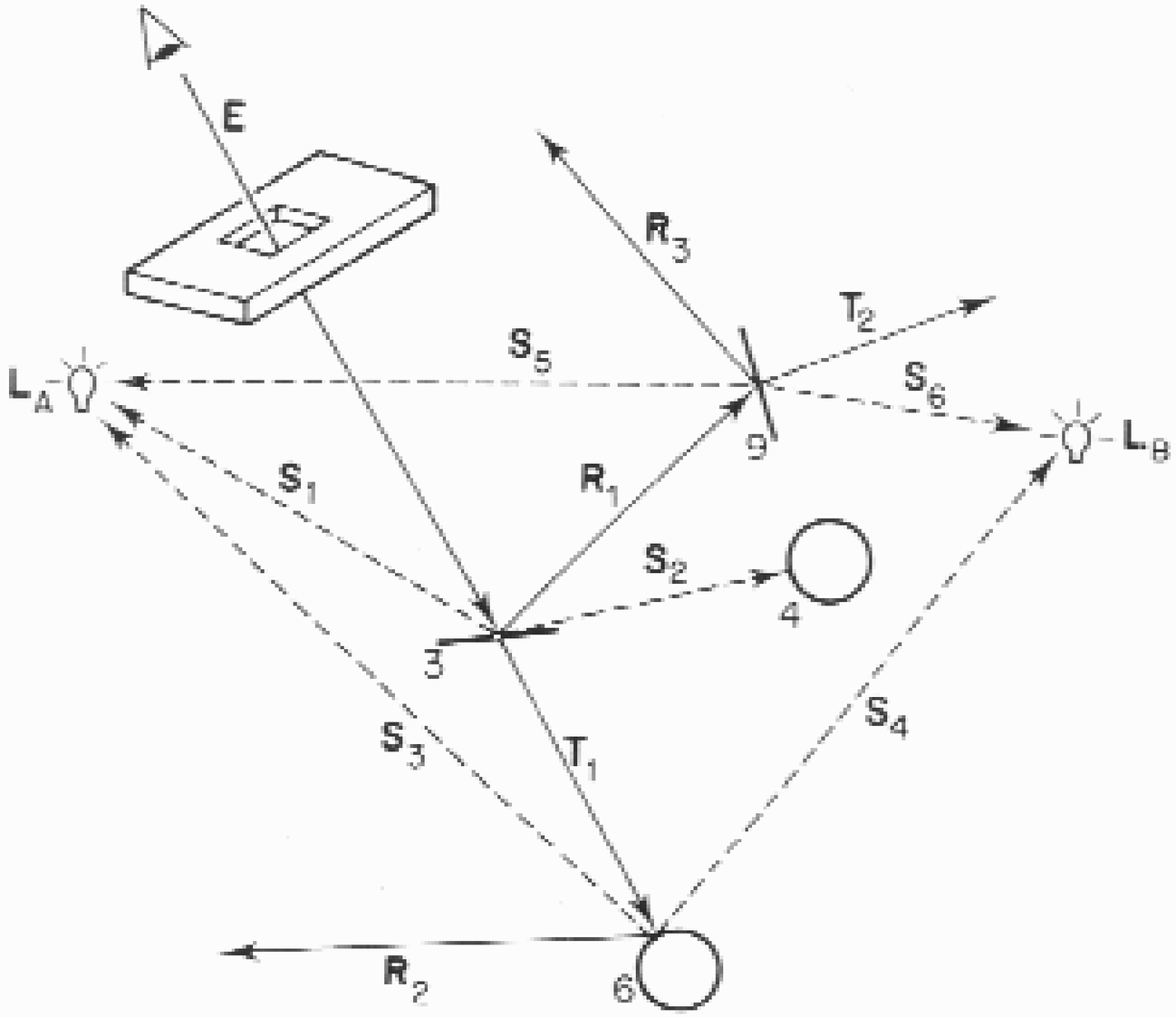
- It only takes into account the rays that arrive to the point of view
- It uses inverse ray tracing
  - from the eye to the light source
- The algorithm depends on the point of view
- It’s a global model with a local calculation for each point

Ray tracing - process

- It calculates the intersection with objects
  - Visibility of the lights
  - It calculates the reflected ray and the refracted ray (or transmitted) for each ray
- It follows until:
  - The ray has low energy
  - The ray goes out of the scene
  - The ray strikes into a diffuse object

Ray tracing - sketch

- Ray tracing from the point of view
  - S: Shadow (to light sources)
  - R: Reflected
  - T: Transmitted

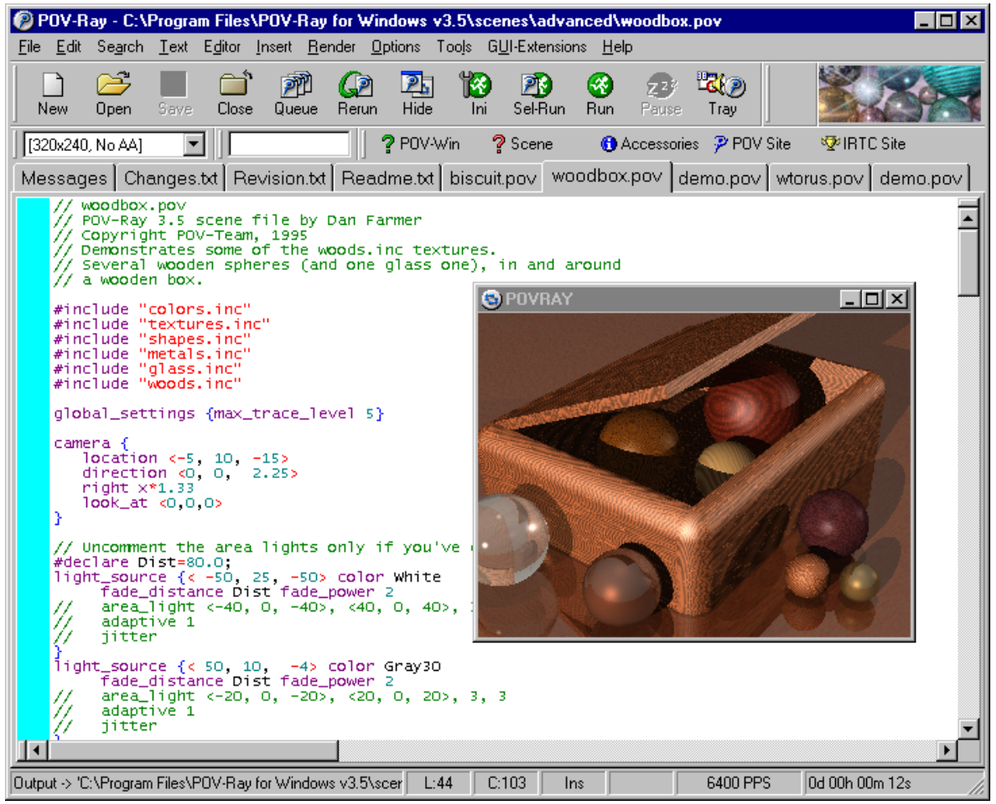


Ray tracing - limitations

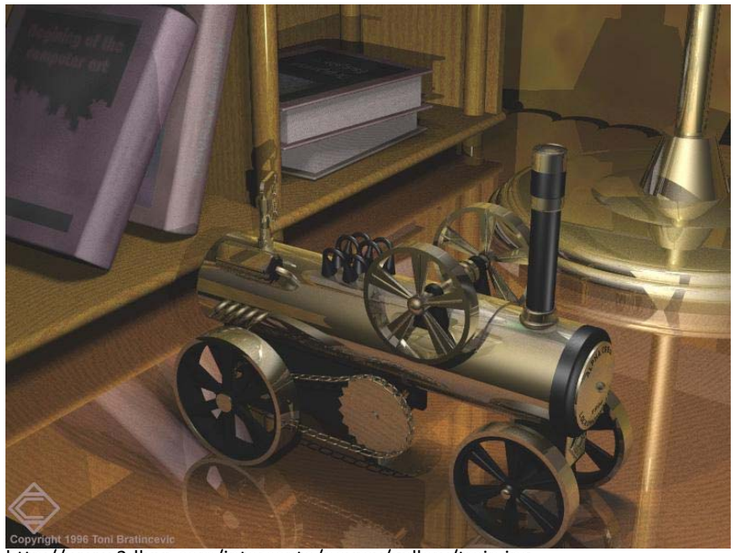
- It only considers the specular reflection and the refraction
- The diffuse reflection is considered only in the ray that arrives from the light source
- It would be very expensive to consider the complete diffuse reflection
- Most of the scenes have surfaces with diffuse reflection

POV-Ray

- Persistence of Vision Raytraces (POV-Ray) is the most known ray tracing application
- It is free:
  www.povray.org
- Easy interface
- It has a visual data editor



Images of PovRay



http://www.3dluvr.com/intercepto/povray/gallery/train.jpg

http://www.xlcus.co.uk/povray/sunset/sunset-0320x0240.jpg

http://www.xlcus.com/povray/tulips/tulips-0240x0320.jpg

http://www.geocities.com/~mloh/povray/2cups.jpg

Radiosity

- It implements the interaction between diffuse surfaces
- It creates a solution independent of the point of view
  - The solution is calculated for all the points of the scene
- It calculates the radiation for each polygon
- It is necessary to discretize (divide in smaller polygons) the scene
  - The discretization process depends on the previous step of the solution

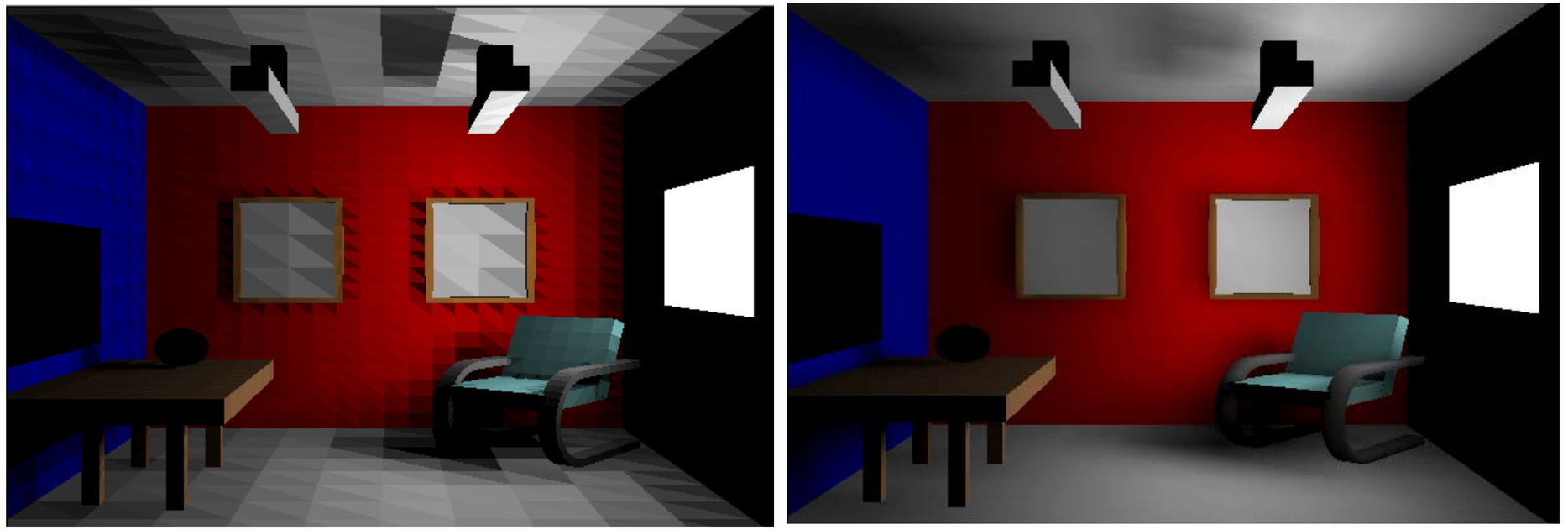
Radiosity - process

- The light sources are considered emitter polygons
- It is calculated the diffuse-diffuse interaction with each visible polygon for the light
- A portion of the light is absorbed and another portion is emitted
- The process follows with the polygon that has the most energy to emit
- The process continues until a defined percentage of the energy has been absorbed

Radiosity – form factor

- The transfer between two polygons is calculated by geometry relations
- The form factors make an average of the radiation transmitted between two polygons
  - It takes into account the visibility between each other

Radiosity – example of process



Radiosity - images

http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_3.htm

Escenario

Radiosity - limitations

- It doesn’t considerer the specular reflection
  - The scenes usually have both reflections
- It is necessary to discretize the scene with polygons before the calculation

Radiance

- It is the “renderer” most known of global illumination (Gregory J. Ward -1994)
- The goal is to represent illumination with the maximum accuracy in architecture images
  - Solar light
  - Artificial light
- It makes separated calculations for specular reflection and diffuse reflection

Radiance - images

Figure 19. Indiana University library space, illuminated by a central skylight.

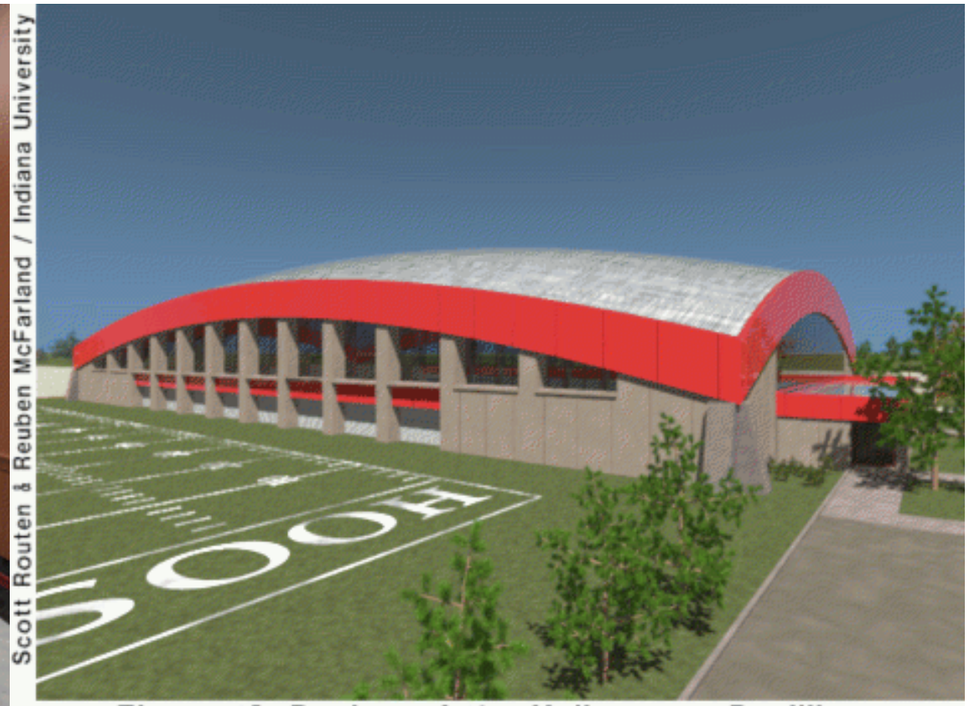


Figure 16. Design of the Mellencamp Pavillion, currently under construction at Indiana University.

http://www.siggraph.org/education/materials/HyperGraph/raytrace/radiance/abstract.html