Graphics, OpenGL, GLUT, GLUI, CUDA, OpenCL, OpenCV, and more!

CS635 Spring 2010

Daniel G. Aliaga
Department of Computer Science
Purdue University

Computer Graphics I

- History and applications
- Computer Graphics Pipeline
- Linear Algebra Review
  - Vectors, points
  - Matrices, transformations
- Representations
  - Points, lines, polygons, objects, meshes
  - Textures and images
- Lighting and Shading
  - Flat, Gouraud, Phong
- Some advanced topics
  - Global illumination
  - Ray tracing
  - Antialiasing

(Some slides courtesy of Thomas Funkhouser and Marcus Magnor)

Computer Graphics II

- OpenGL
  - Motivation
  - Graphics context/state
  - Basic program outline
- Rendering geometric primitives
  - Points, lines, polygons
- Lighting and Shading
  - Flat, Gouraud, Phong
- Texturing Polygons
- GLUT
- GLUI
- CUDA and OpenCL
- OpenCV

History

- 1950: MIT Whirlwind (CRT)
- 1955: Sage, Radar with CRT and light pen
- 1958: Willy Higinbotham “Tennis”
- 1960: MIT “Spacewar” on DEC-PDP-1
- 1963: Ivan Sutherland’s “Sketchpad” (CAD)
- 1968: Tektronix storage tube
- 1968: Evans & Sutherland’s flight simulators
- 1968: Douglas Engelbart: computer mouse
- 1969: ACM SIGGRAPH
- 1970: Xerox GUI
- 1971: Gouraud shading
- 1974: Z-buffer
- 1975: Phong Model
- 1979: Eurographics
- 1981: Apollo Workstation, PC
- 1982: Whitted: Ray tracing
- 1982: SGI
- 1984: X Window System
- 1984: 1st SGI Workstation
- ->1995: SGI dominance
- ->2003: PC dominance
- Today: programmable graphics hardware (again)

Applications

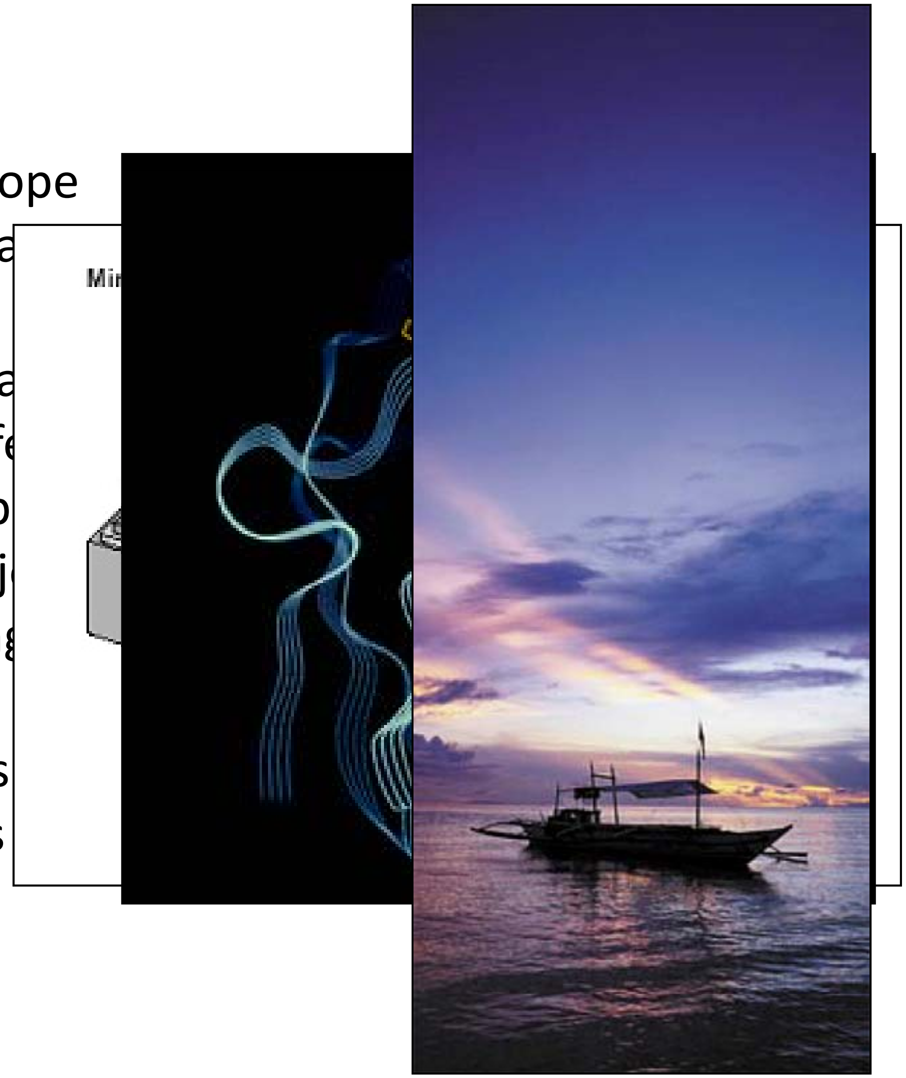
- Training
- Education
- Computer-aided design (CAD)
- Scientific Visualization
- E-commerce
- Computer art
- Entertainment

Ivan Sutherland (1963) - SKETCHPAD

- pop-up menus
- constraint-based drawing
- hierarchical modeling

Display hardware

- vector displays
  - 1963 – modified oscilloscope
  - 1974 – Evans and Sutherla
- raster displays
  - 1975 – Evans and Sutherla
  - 1980s – cheap frame buffe
  - 1990s – liquid-crystal disp
  - 2000s – micro-mirror proj
  - 2010s – high dynamic rang
- other
  - stereo, head-mounted dis
  - autostereoscopic displays



Input hardware

- 2D
  - light pen, tablet, mouse, joystick, track ball, touch panel, etc.
  - 1970s & 80s - CCD analog image sensor + frame grabber

Input hardware

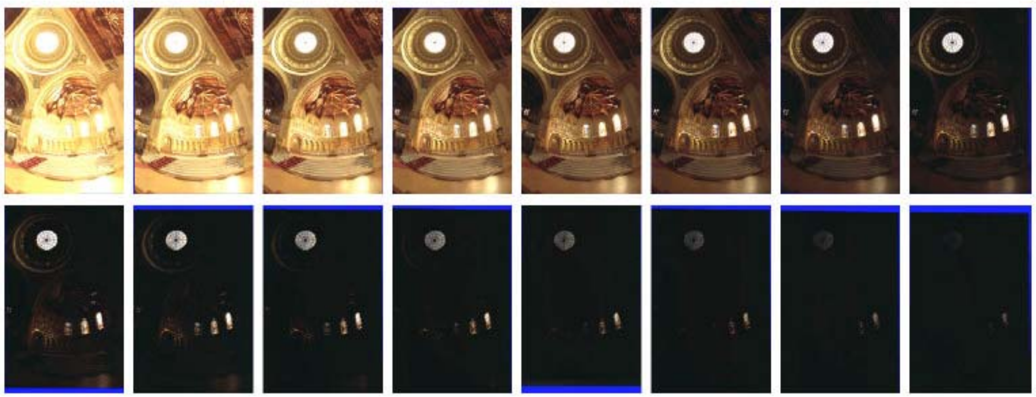
- 2D
  -
  -

Input hardware

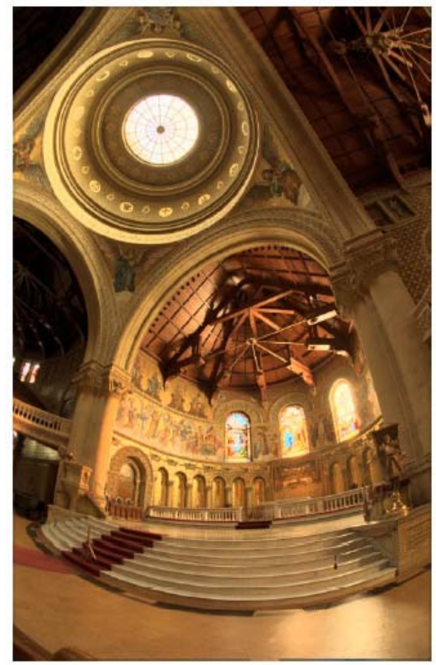
- 2D
  - light pen, tablet, mouse, joystick, track ball, touch panel, etc.
  - 1970s & 80s - CCD analog image sensor + frame grabber
  - 1990s & 2000’s - CMOS digital sensor + in-camera processing

High Dynamic Range Imaging

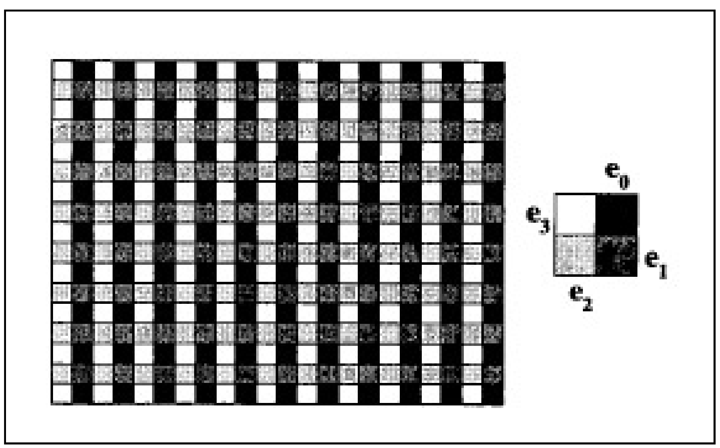
- negative film = 130:1 (7 stops)
- paper prints = 46:1
- combine multiple exposures = 250,000:1 (18 stops)



→



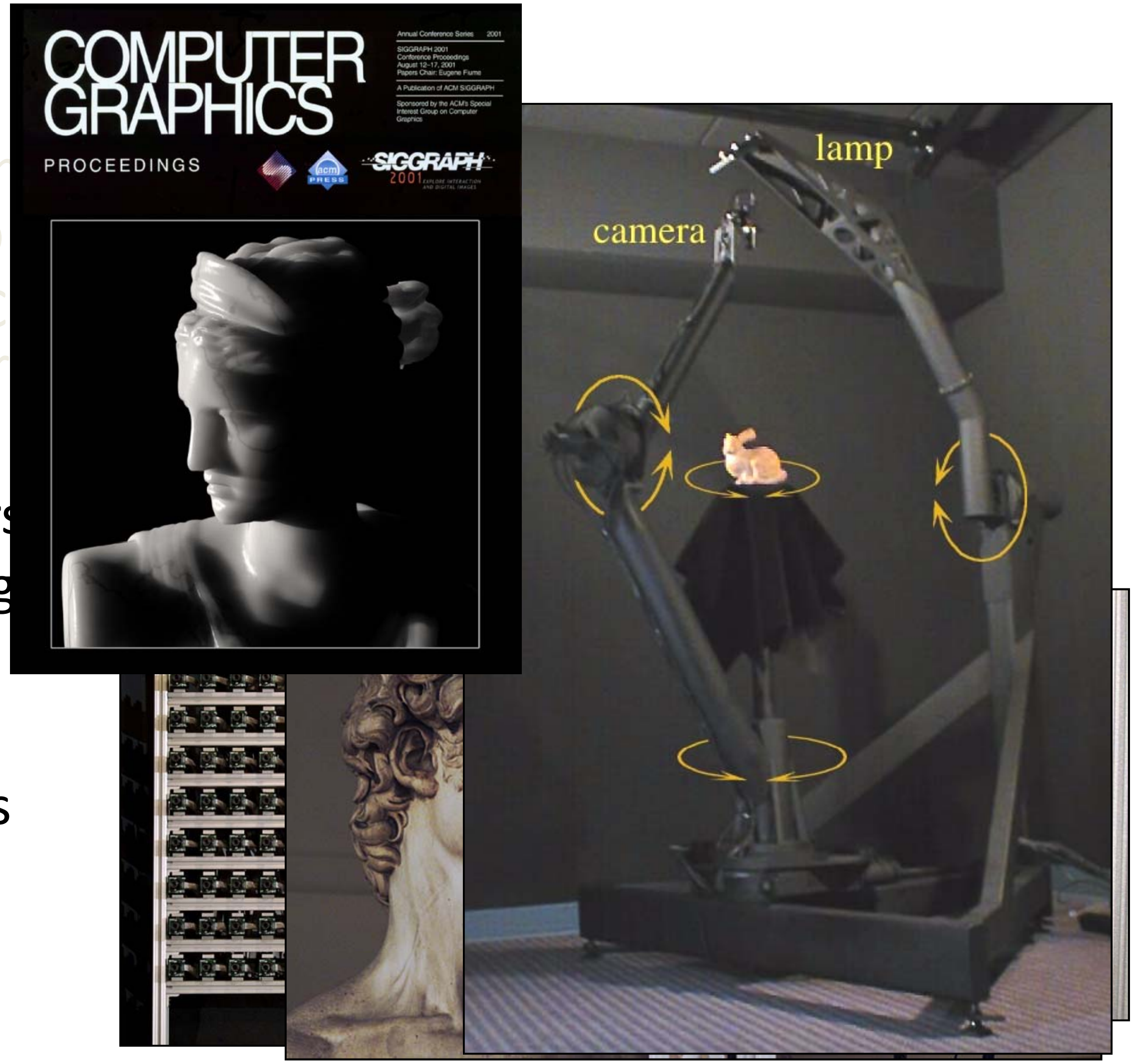
[Debevec97]



[Nayar00]

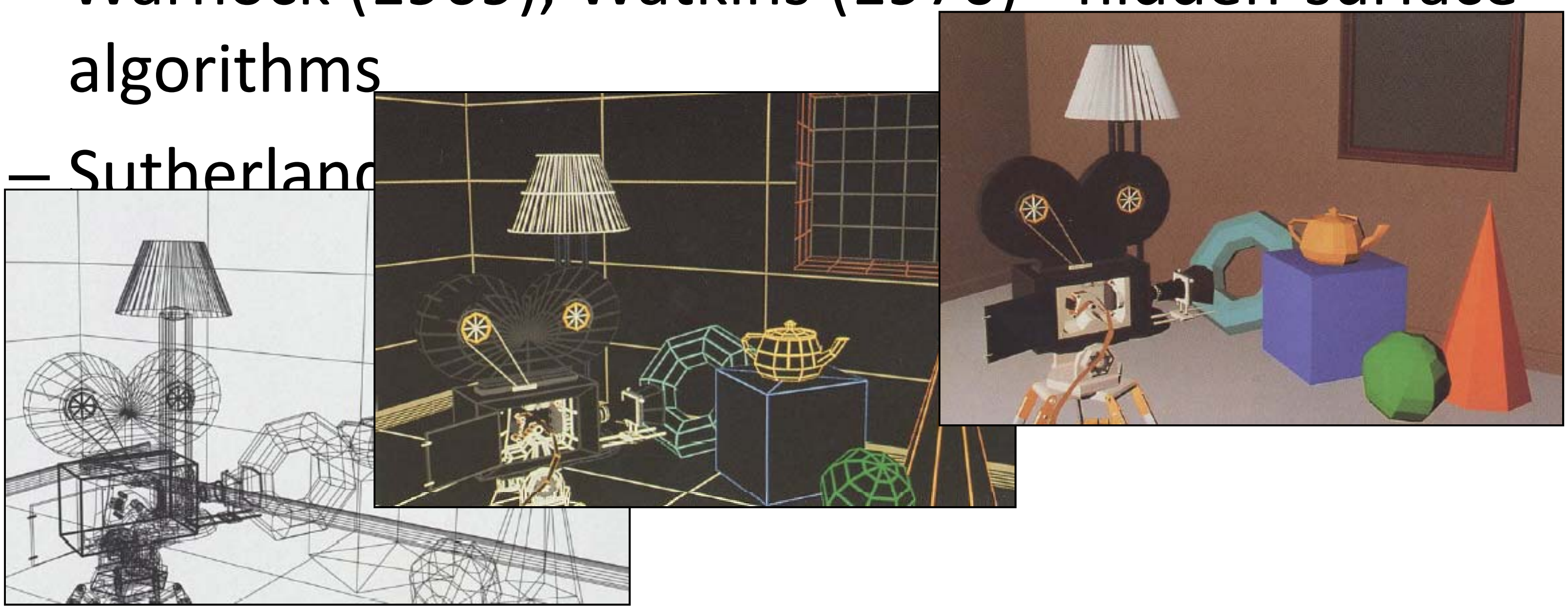
Input hardware

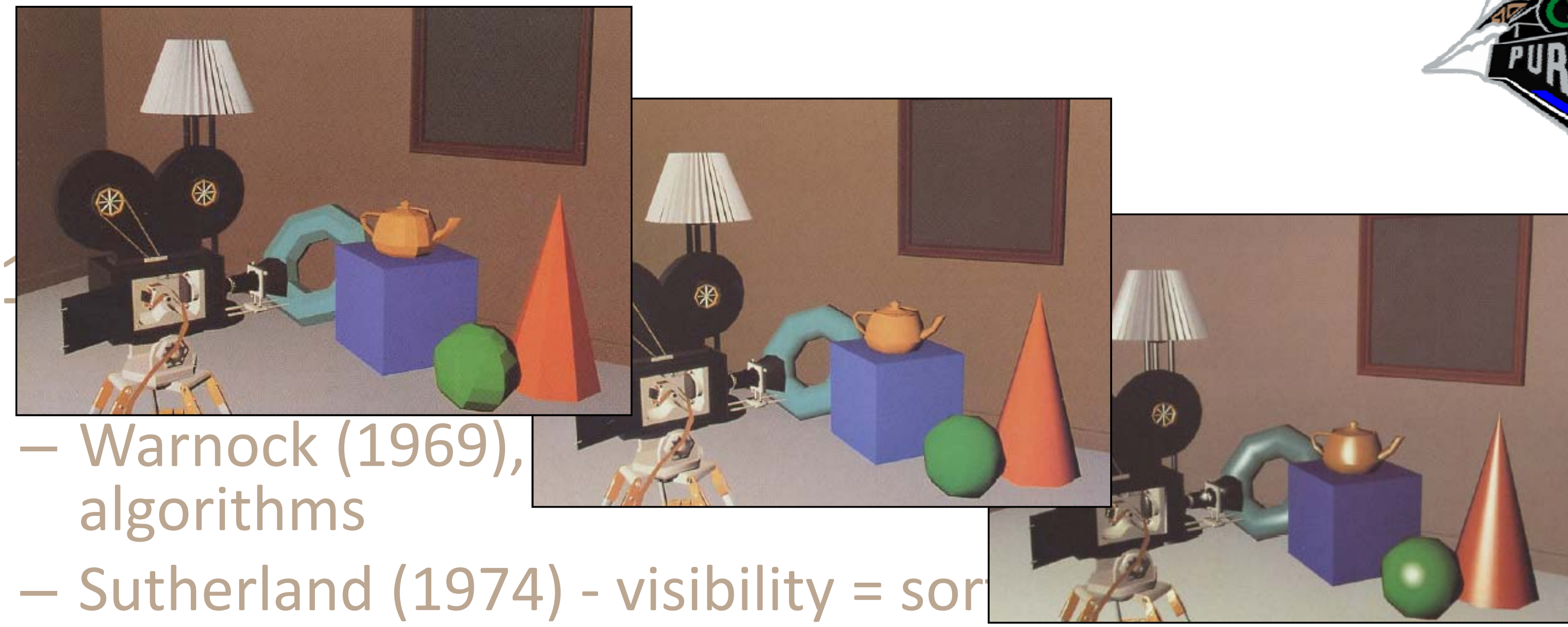
- 2D
  - light pen, tablet, m
  - 1970s & 80s - CCD
  - 1990s & 2000's - C
    → high-dynamic r
- 3D
  - 1980s - 3D trackers
  - 1990s - active rang
- 4D and higher
  - multiple cameras
  - multi-arm gantries



Rendering

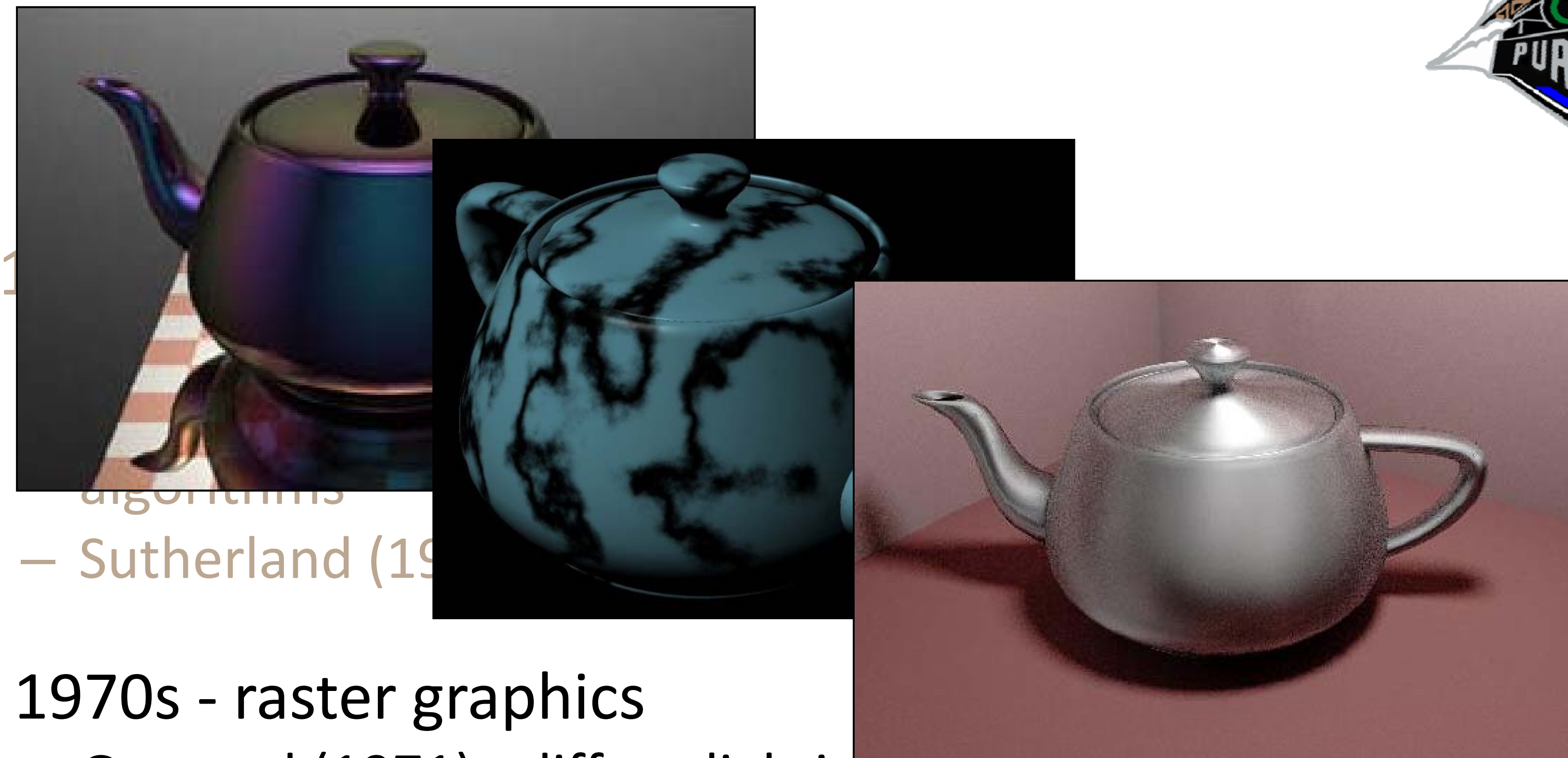
- 1960s - the visibility problem
  - Roberts (1963), Appel (1967) - hidden-line algorithms
  - Warnock (1969), Watkins (1970) - hidden-surface algorithms
  - Sutherland





- 1
  - Warnock (1969), algorithms
  - Sutherland (1974) - visibility = sor

- 1970s - raster graphics
  - Gouraud (1971) - diffuse lighting
  - Phong (1974) - specular lighting
  - Blinn (1974) - curved surfaces, texture
  - Crow (1977) - anti-aliasing

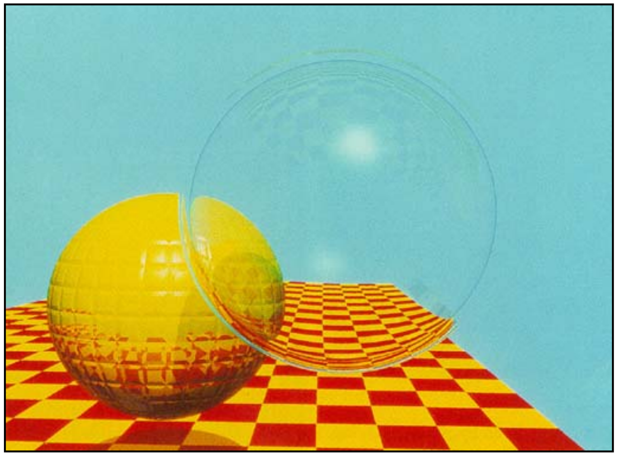


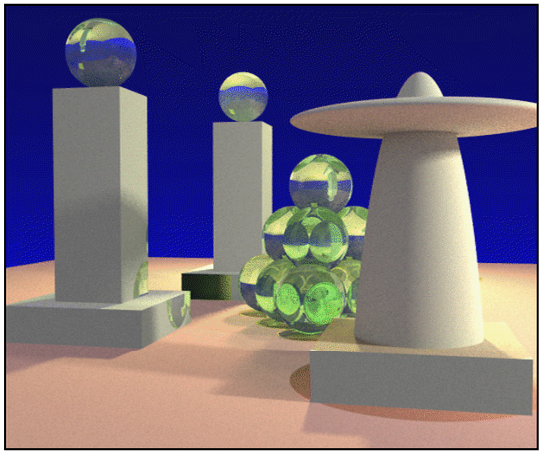
algorithms

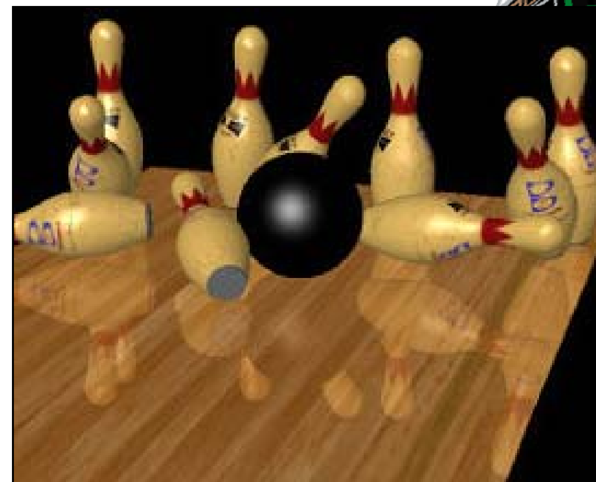
– Sutherland (19

- 1970s - raster graphics
  - Gouraud (1971) - diffuse lighting
  - Phong (1974) - specular lighting
  - Blinn (1974) - curved surfaces, texture
  - Catmull (1974) - Z-buffer hidden-surface algorithm
  - Crow (1977) - anti-aliasing

- early 1980s - global illumination
  - Whitted (1980) - ray tracing
  - Goral, Torrance et al. (1984), Cohen (1985) - radiosity
  - Kajiya (1986) - the rendering equation





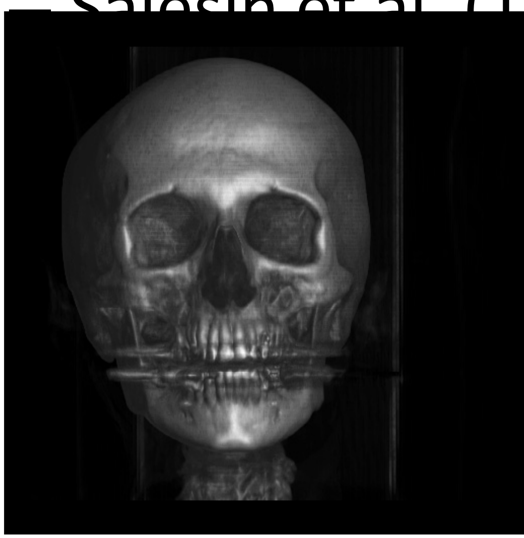
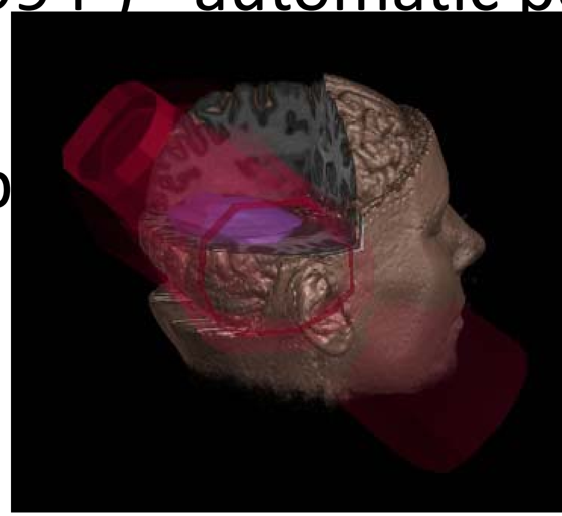
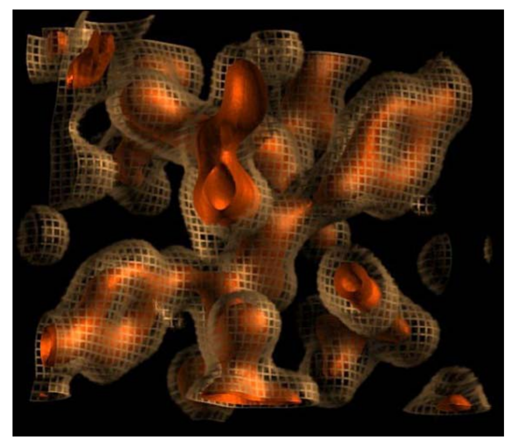


- e ... pa ... n
  - r ...
  - Goral, Torrance et ... en (1985) - radiosity
  - Kajiya (1986) - the ... ation

- late 1980s - photorealism
  - Cook (1984) - shade trees
  - Perlin (1985) - shading languages
  - Hanrahan and Lawson (1990) - RenderMan

- early 1990s - non-photorealistic rendering
  - Drebin et al. (1988), Levoy (1988) - volume rendering
  - Haeberli (1990) - impressionistic paint programs
  - Salesin et al. (1994-) - automatic pen-and-ink

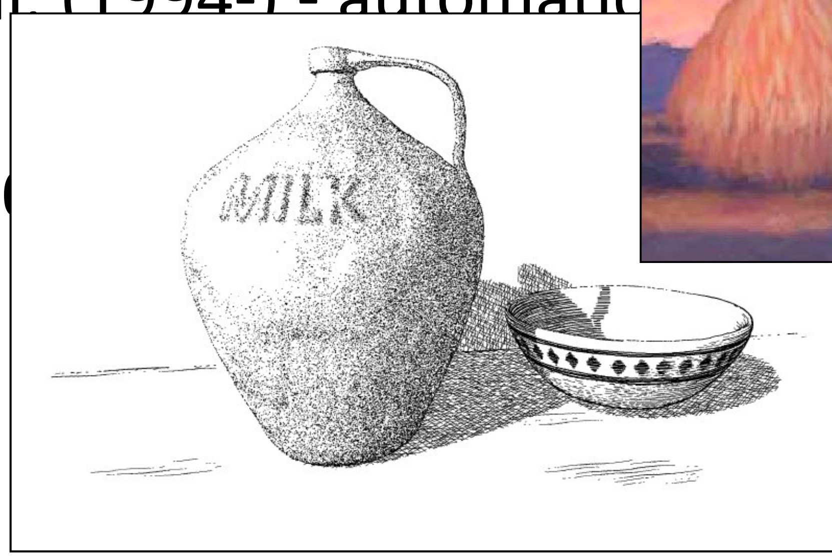
p

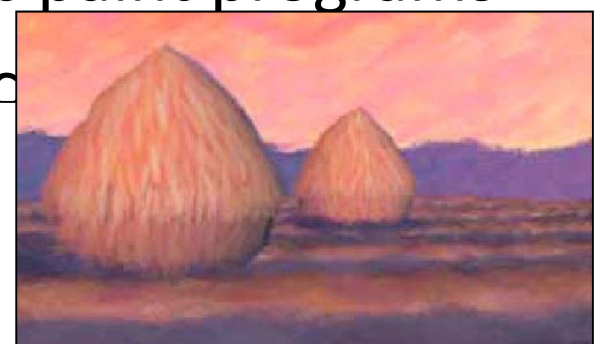




- early 1990s - non-photorealistic rendering
  - Drebin et al. (1988), Levoy (1988) - volume rendering
  - Haeberli (1990) - impressionistic paint programs
  - Salesin et al. (1994-) - automatic

n

9



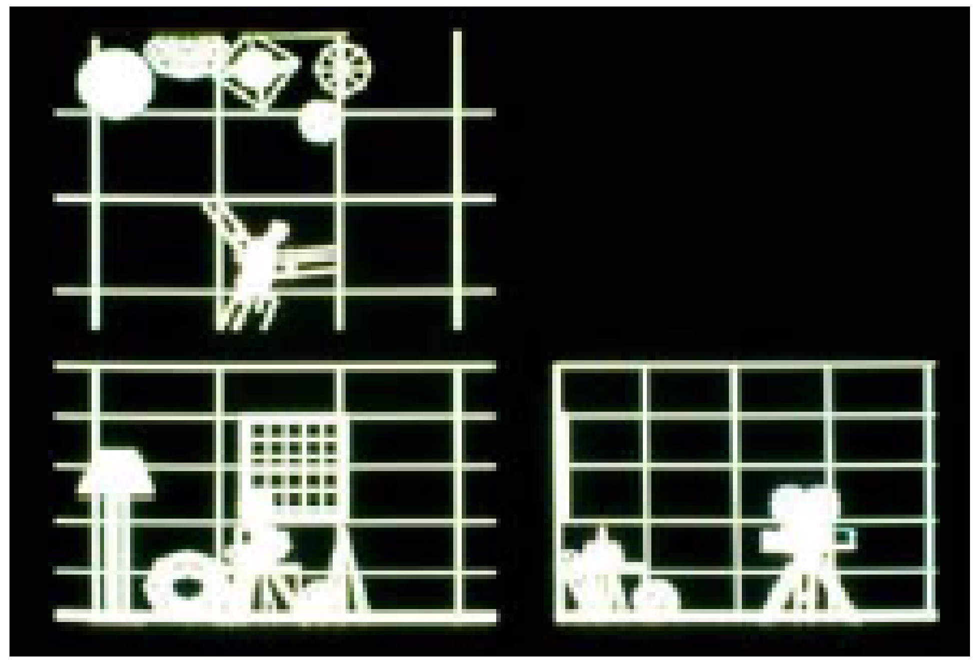


Computer Graphics Pipeline

- How do we create a rendering such as this?

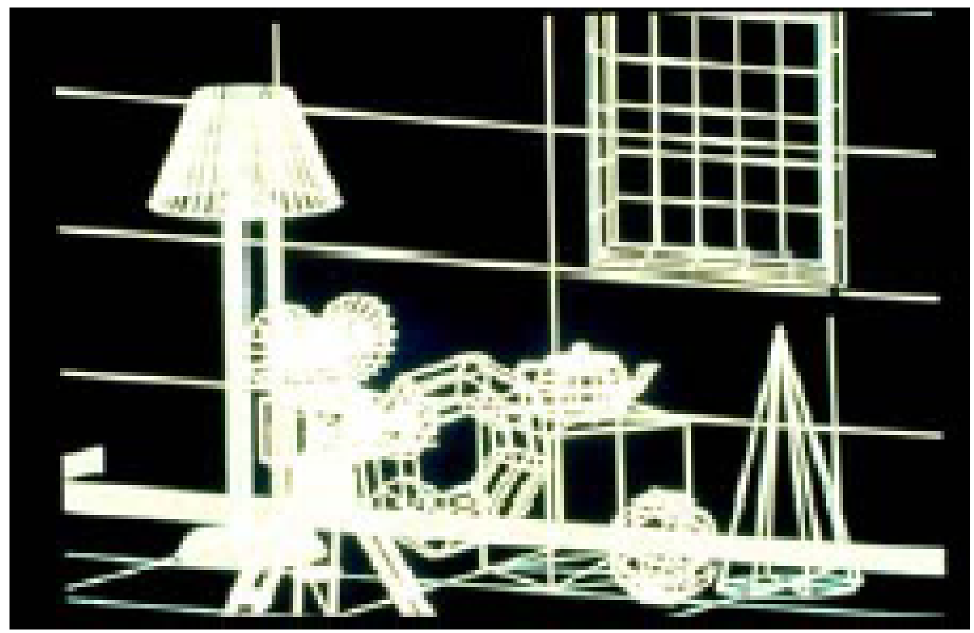
Computer Graphics Pipeline

- Design the scene (technical drawing in “wireframe”)



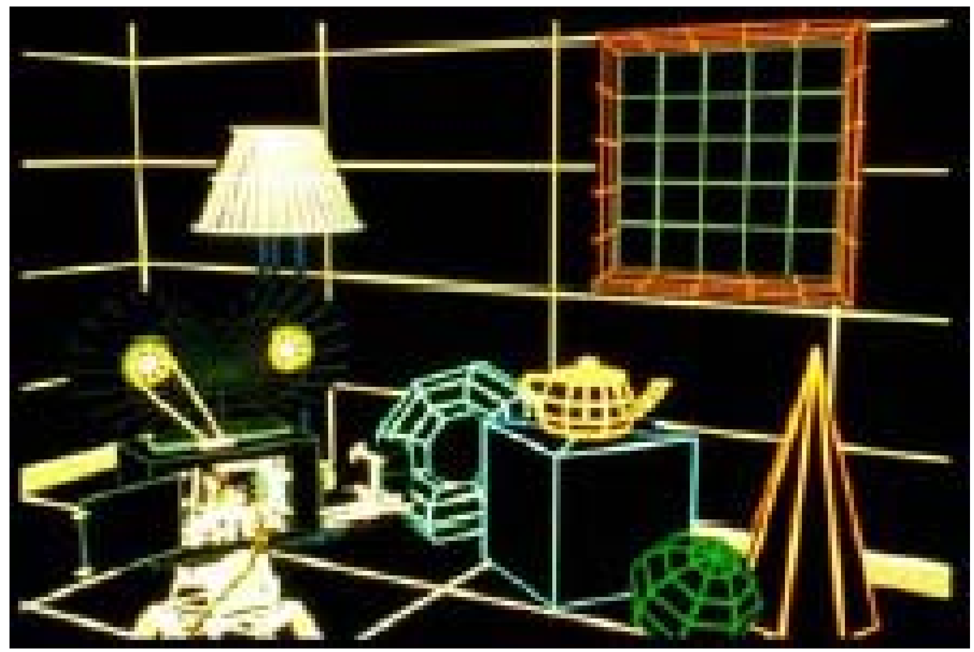
Computer Graphics Pipeline

- Apply perspective transformations to the scene geometry for a virtual camera



Computer Graphics Pipeline

- Hidden lines removed and colors added



Computer Graphics Pipeline

- Geometric primitives filled with constant color

Computer Graphics Pipeline

- View-independent lighting model added

Computer Graphics Pipeline

- View-dependent lighting model added

Computer Graphics Pipeline

- Texture mapping: pictures are wrapped around objects

Computer Graphics Pipeline

- Reflections, shadows, and bumpy surfaces

Computer Graphics Pipeline

Geometric Primitives

Modeling Transformation	Transform into 3D world coordinate system
Lighting	Simulate illumination and reflectance
Viewing Transformation	Transform into 3D camera coordinate system
Clipping	Clip primitives outside camera's view
Projection Transformation	Transform into 2D camera coordinate system
Scan Conversion	Draw pixels (incl. texturing, hidden surface…)

Image

Linear Algebra Review

- Why do we need it?
  - Modeling transformation
    - Move “objects” into place relative to a world origin
  - Viewing transformation
    - Move “objects” into place relative to camera
  - Perspective transformation
    - Project “objects” onto image plane

Transformations

- Most popular transformations in graphics
  - Translation
  - Rotation
  - Scale
  - Projection
- In order to use a single matrix for all, we use homogeneous coordinates…

Transformations

Transformations

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

Identity

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Translation

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

Mirror over X axis

Transformations

Rotate around Z axis:

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

Rotate around Y axis:

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

Rotate around X axis:

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

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspective projection

Representations

- How are the objects described in a computer?
  - Points (or vertices)
  - Lines
  - Triangles
  - Polygons
  - Curved surfaces, etc.
  - Functions

Representations

- How are the geometric primitives grouped?
  - "Polygon soup"
  - Vertex-array/triangle-strip
  - Mesh

Representations

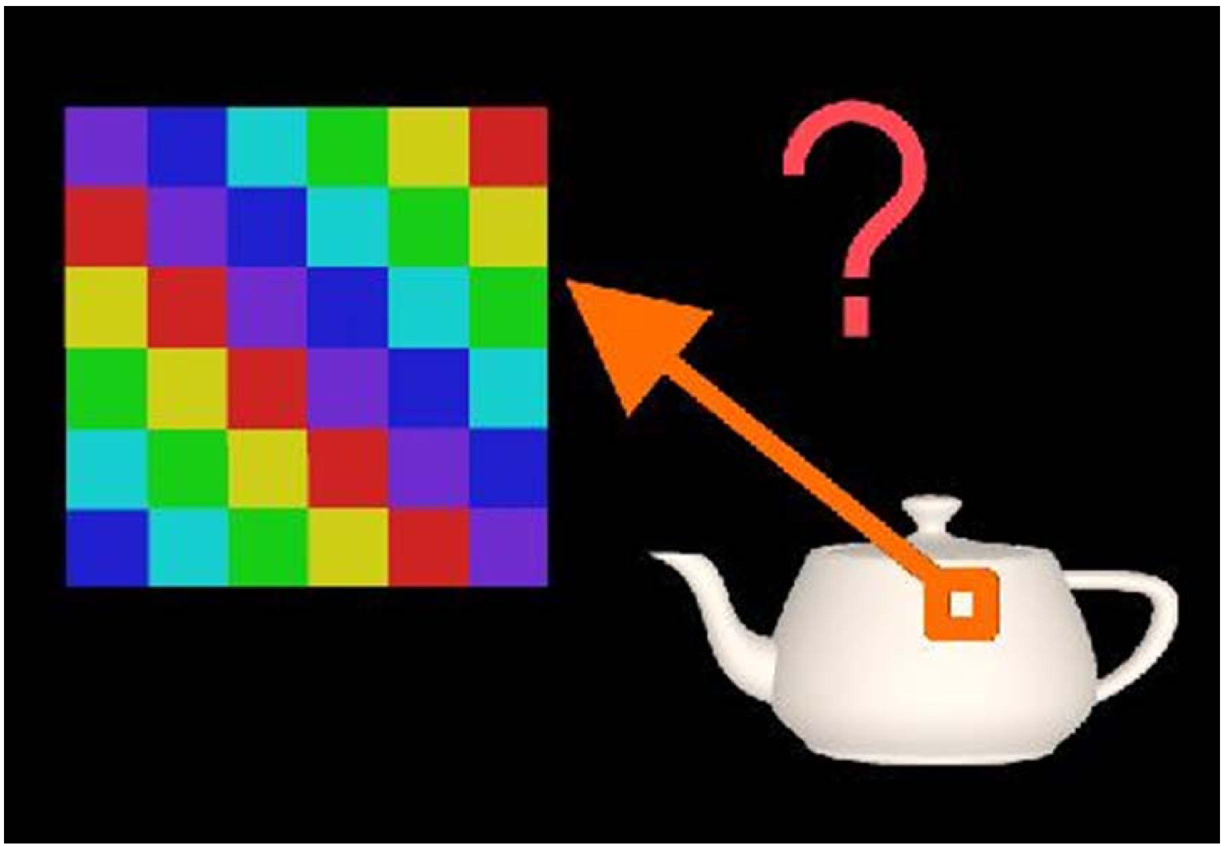
- What information is needed per vertex?
  - Position
  - Normal
  - Color
  - Texture coordinates…

Representations

- What information is needed per geometric primitive?
  - Color
  - Normal
  - Material properties (e.g. textures…)

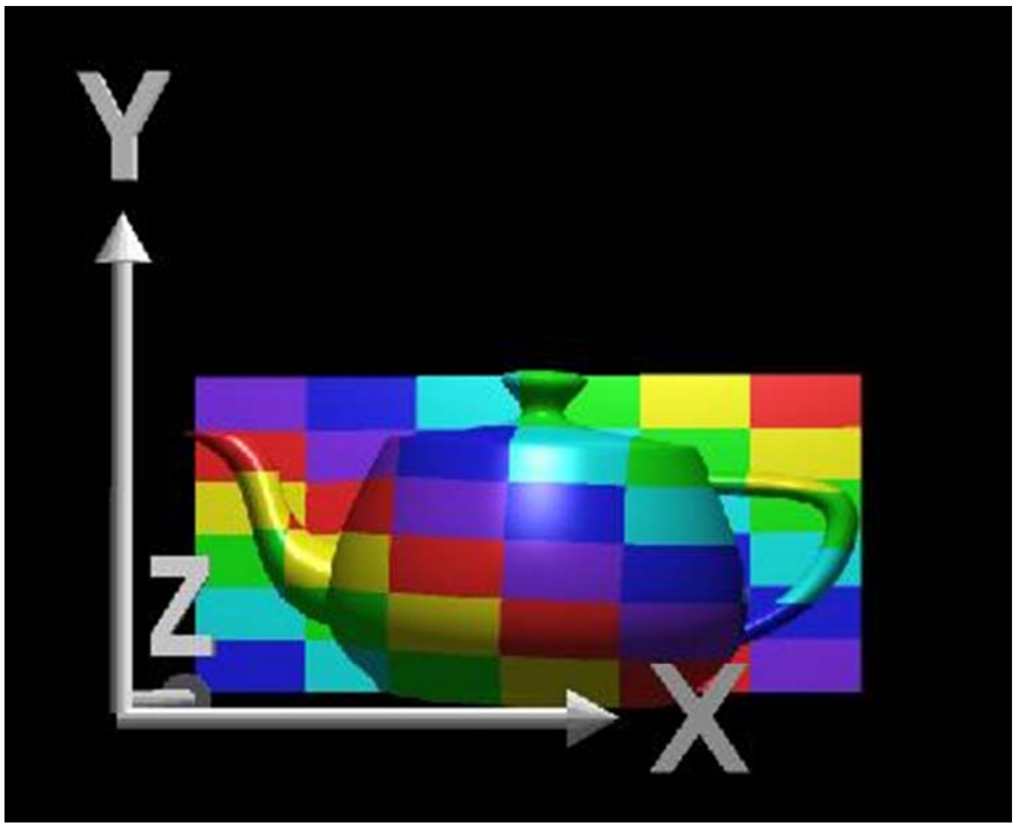
Texture Mapping

- Map a “texture” onto the surface of an object
  - Wood, marble, or any “pattern”

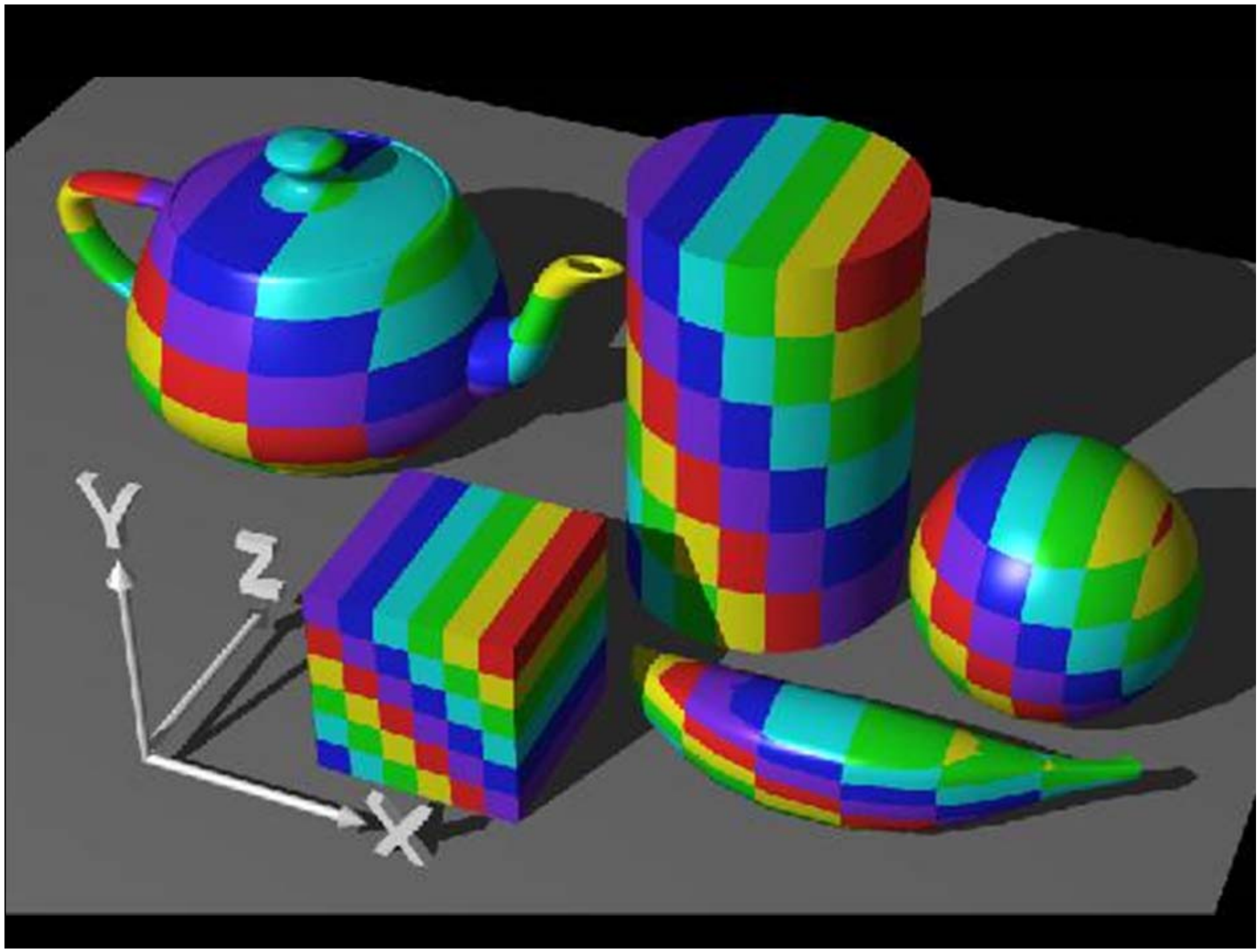


Texture Mapping

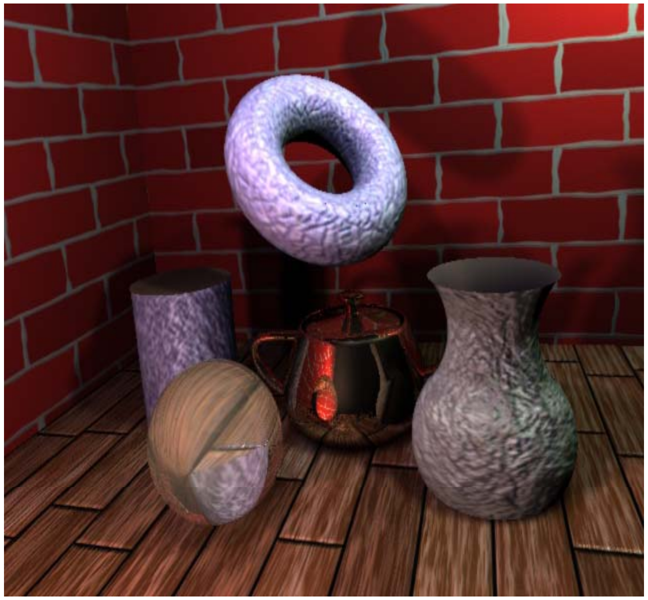
- A texture is a two-dimensional array of “texels”, indexed by a (u,v) texture coordinate
- At each screen pixel, a texel can be used to substitute a geometric primitives surface color



Texture Mapping



Texture Mapping



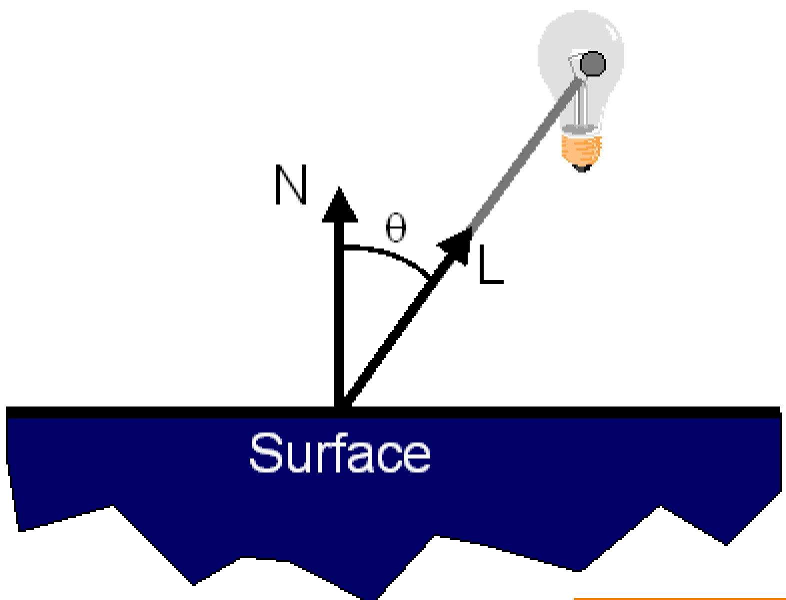
Lighting and Shading

- Light sources
  - Point light
    - Models an omnidirectional light source (e.g., a bulb)
  - Directional light
    - Models an omnidirectional light source at infinity
  - Spot light
    - Models a point light with direction

- Light model
  - Ambient light
  - Diffuse reflection
  - Specular reflection

Lighting and Shading

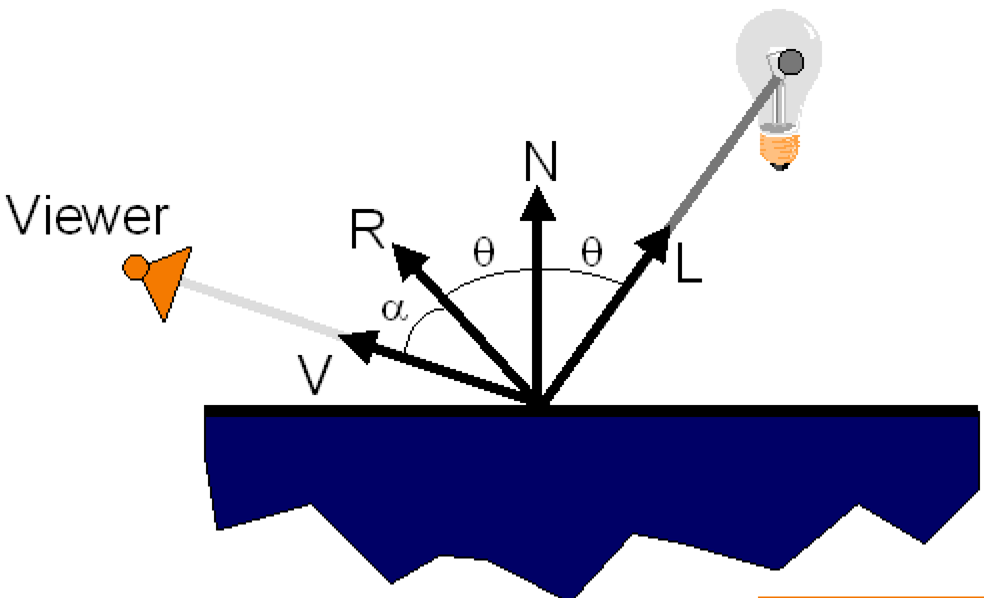
- Diffuse reflection
  - Lambertian model



$$I_D = K_D(N \bullet L)I_L$$

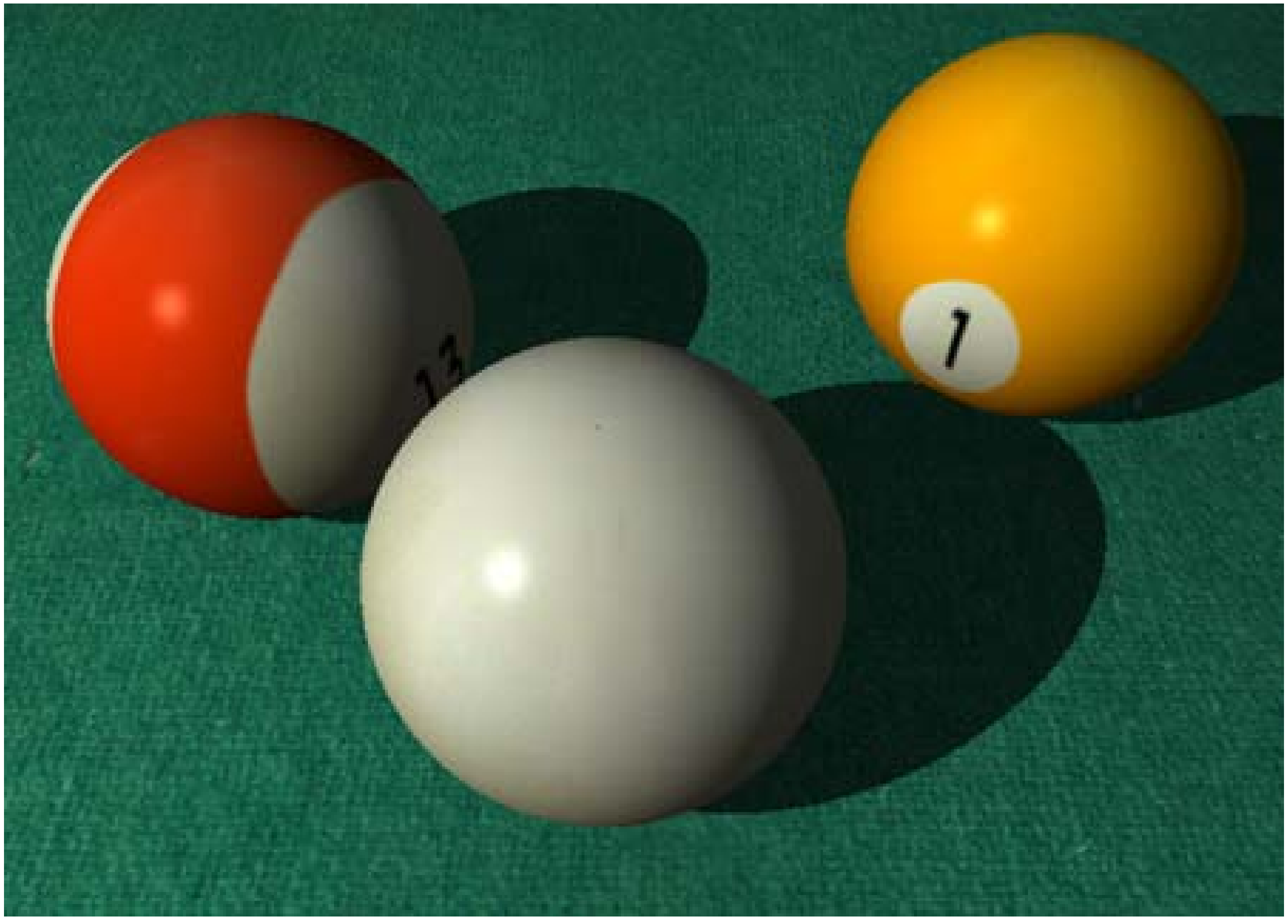
Lighting and Shading

- Specular reflection
  - Phong model

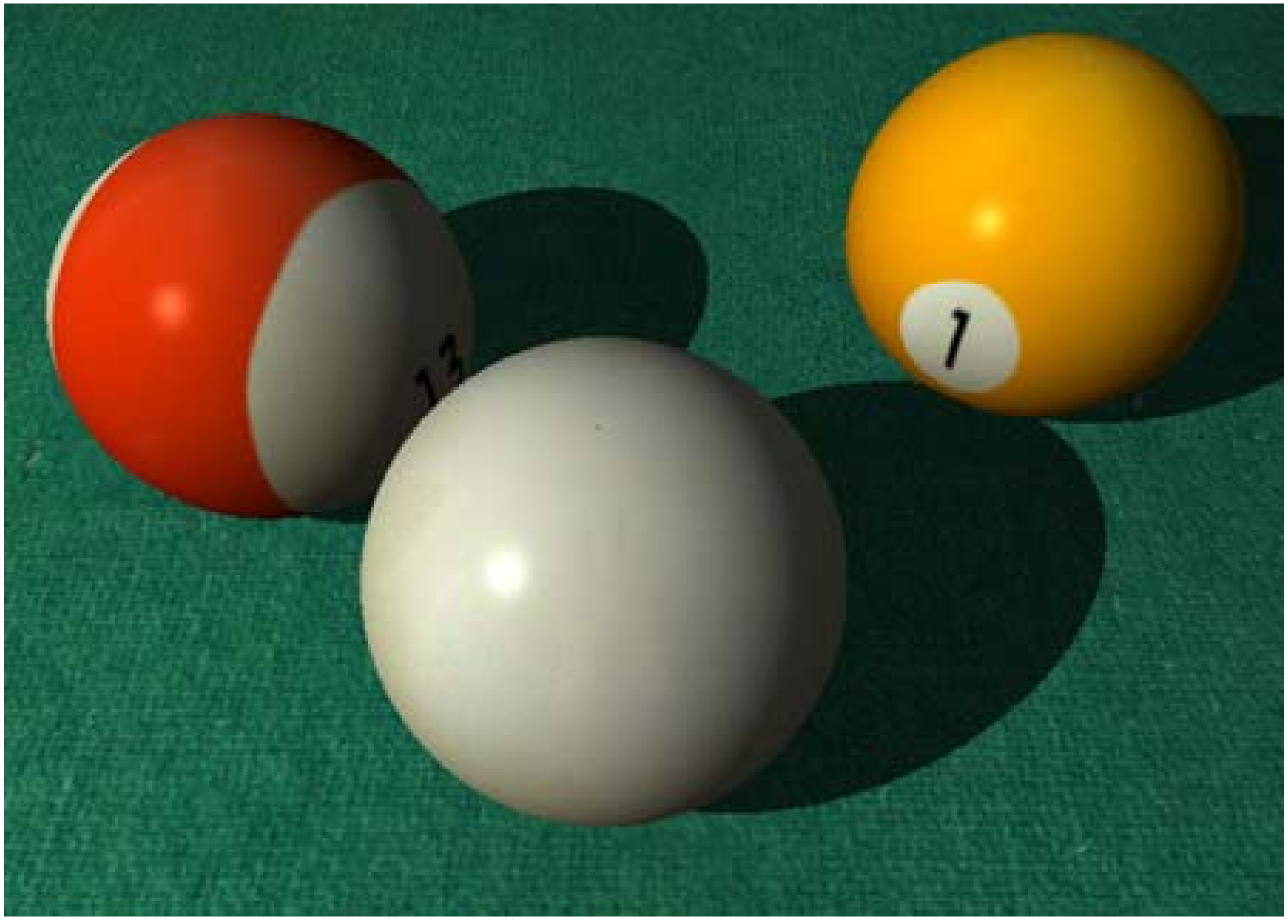


$$I_S = K_S (V \bullet R)^n I_L$$

Lighting and Shading

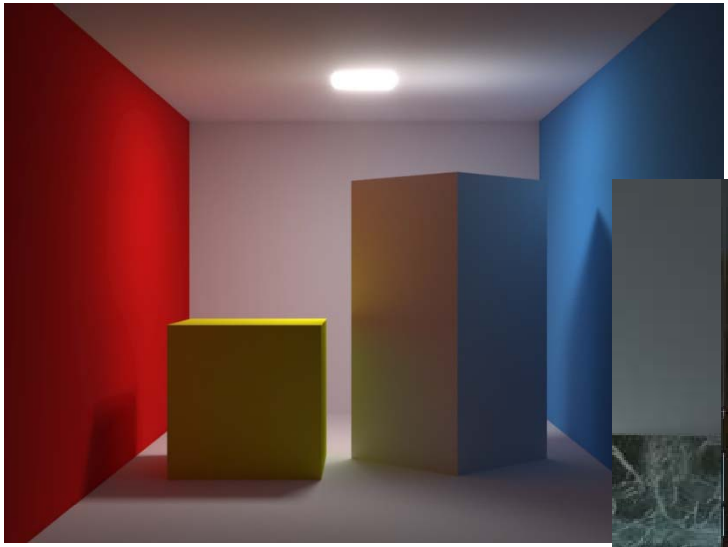


Lighting and Shading

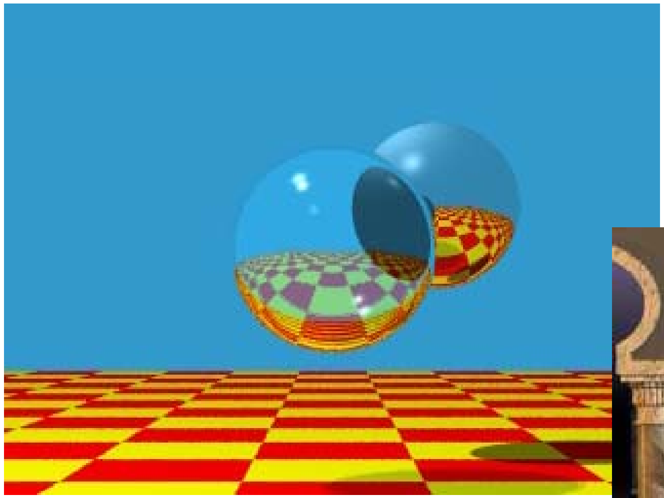


…shadows?

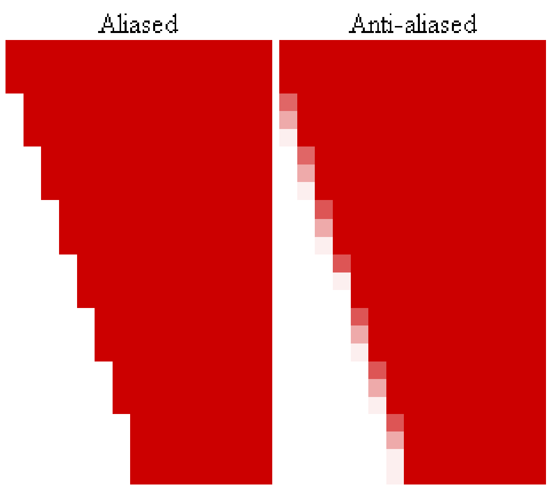
Advanced Topics: Global Illumination

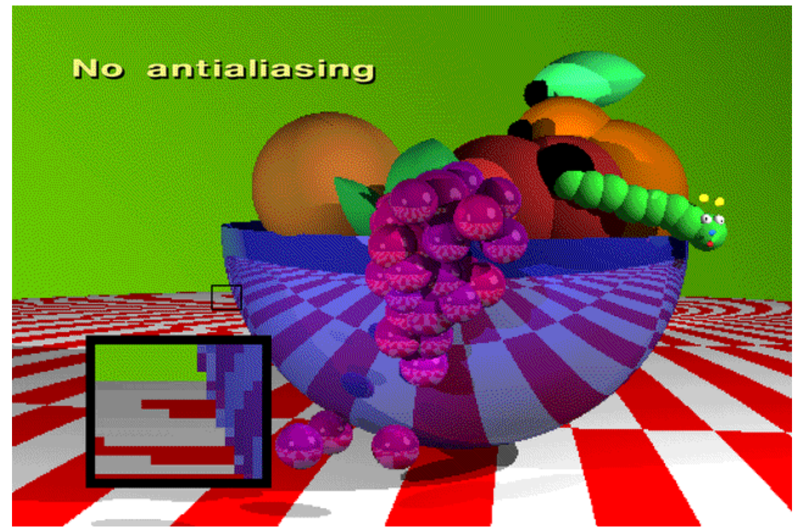


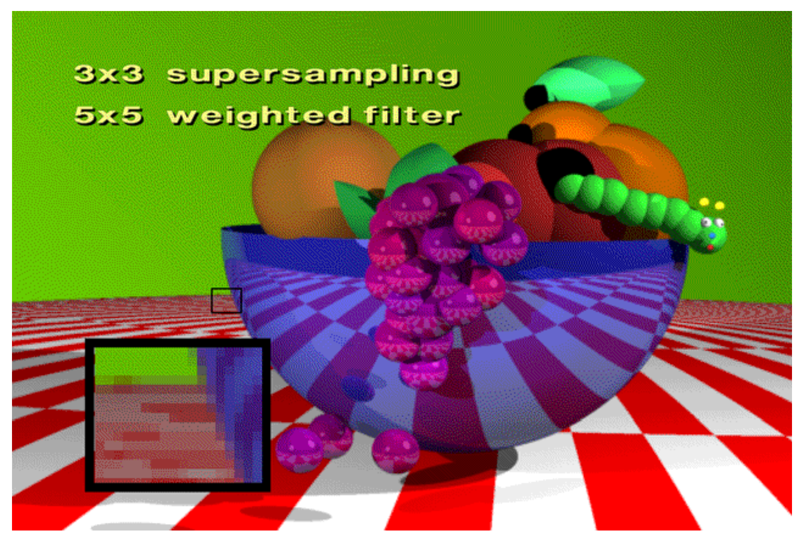
Advanced Topics: Ray tracing



Advanced Topics: Antialiasing







OpenGL

- Software interface to graphics hardware
- ~150 distinct commands
- Hardware-independent and widely supported
  - To achieve this, no windowing tasks are included
- GLU (Graphics Library Utilities)
  - Provides some higher-level modeling features such as curved surfaces, objects, etc.
- Open Inventor (old)
  - A higher-level object-oriented software package

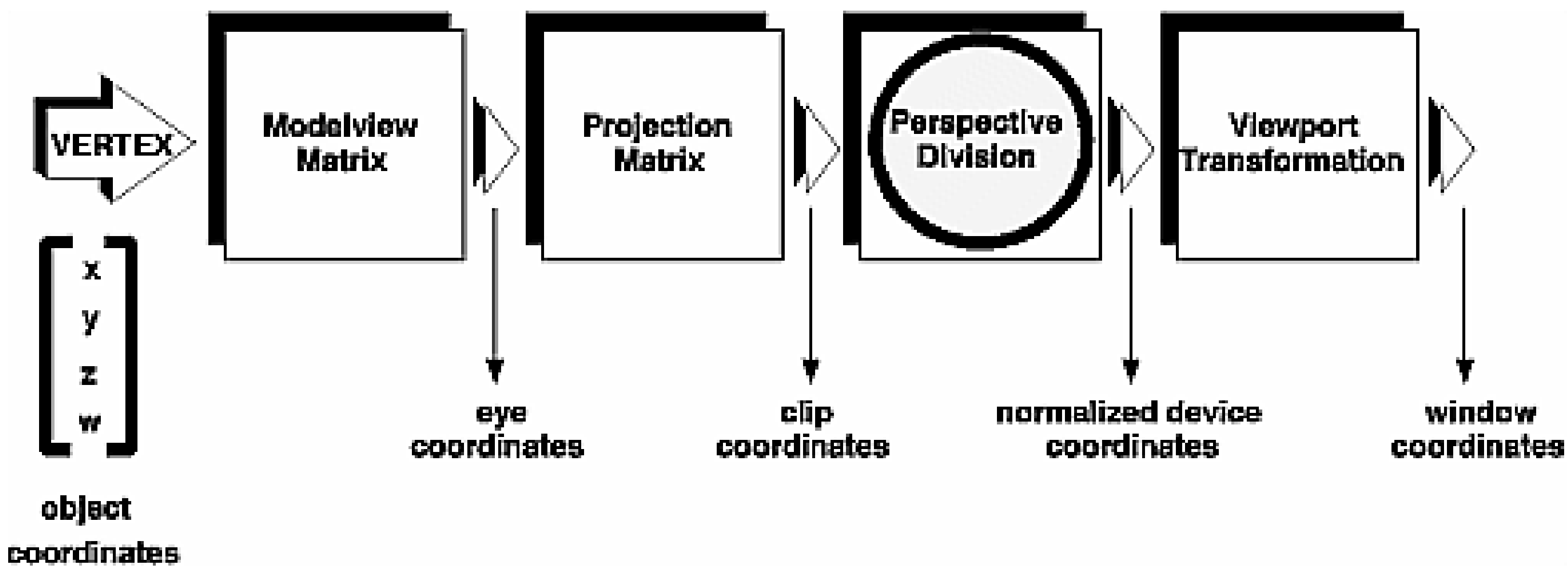
OpenGL Online

- Programming Guide v1.1 ("Red book")
  - http://www.glprogramming.com/red/
- Reference Manual v1.1 ("Blue book")
  - http://www.glprogramming.com/blue/
- Current version is >2.0

OpenGL

- Rendering parameters
  - Lighting, shading, lots of little details…
- Texture information
  - Texture data, mapping strategies
- Matrix transformations
  - Projection
  - Model view
  - (Texture)
  - (Color)

Matrix Transformations



Matrix Transformations

- Each of modelview and projection matrix is a 4x4 matrix
- OpenGL functions
  - glMatrixMode(…)
  - glLoadIdentity(…)
  - glLoadMatrixf(…)
  - glMultMatrix(…)
  - glTranslate(…)
  - glScale(…)
  - glRotate(…)

  - glPushMatrix()
  - glPopMatrix()

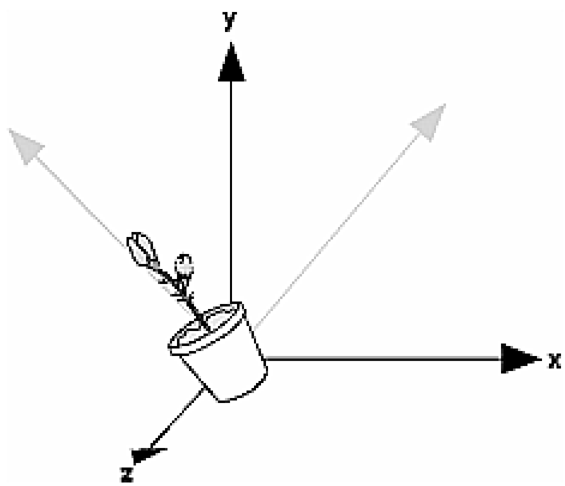
Matrix Transformations

```
{
    …
    …
    …
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glMultMatrixf(N); /* apply transformation */
    glMultMatrixf(M); /* apply transformation M */
    glMultMatrixf(L); /* apply transformation L */
    glBegin(GL_POINTS);
     glVertex3f(v); /* draw transformed vertex v */
    glEnd();
    …
    …
    …
}
```

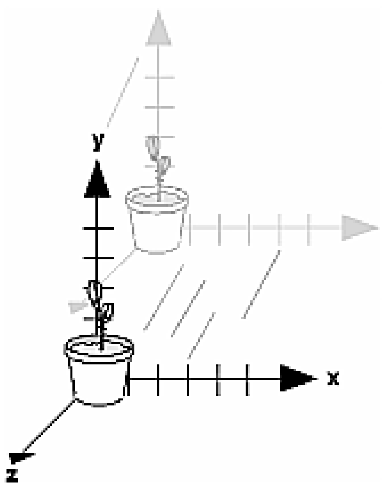
= draw transformed point "N(M(Lv))"

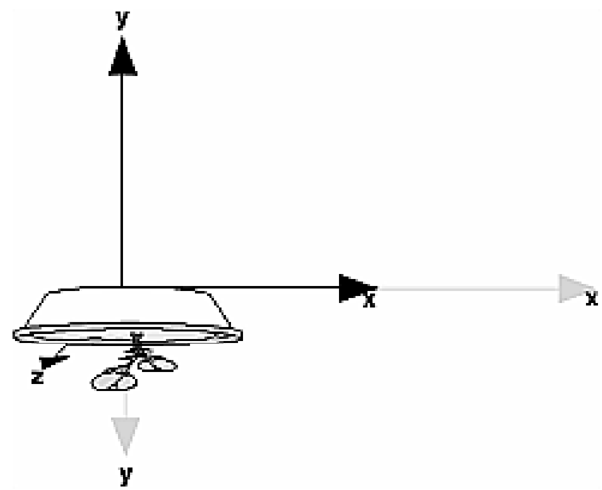
Modelview Transformations

glRotatef(45,0,0,1)



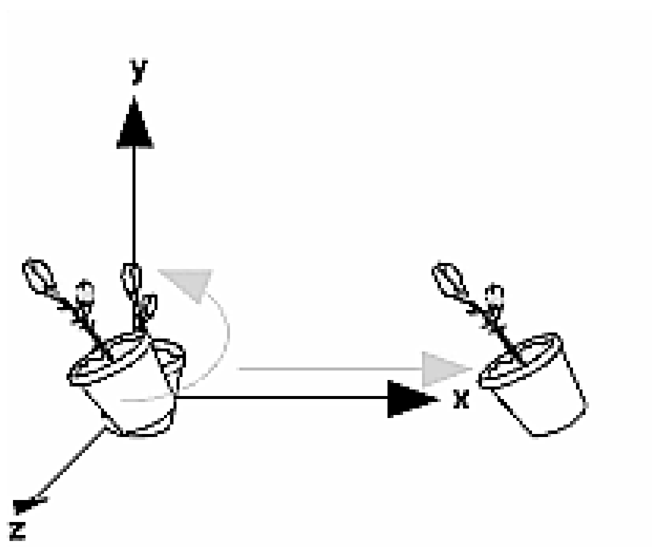
glTranslate3f(tx,ty,tz)



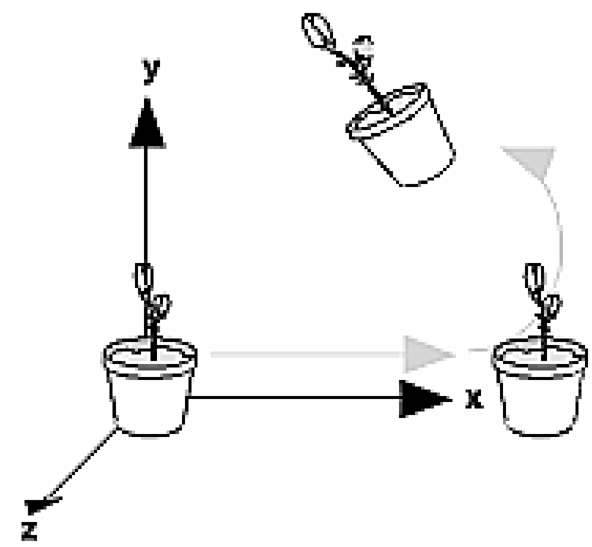


glScalef(2,-0.5,1.0)

Modelview Transformations



Rotate then Translate



Translate then Rotate

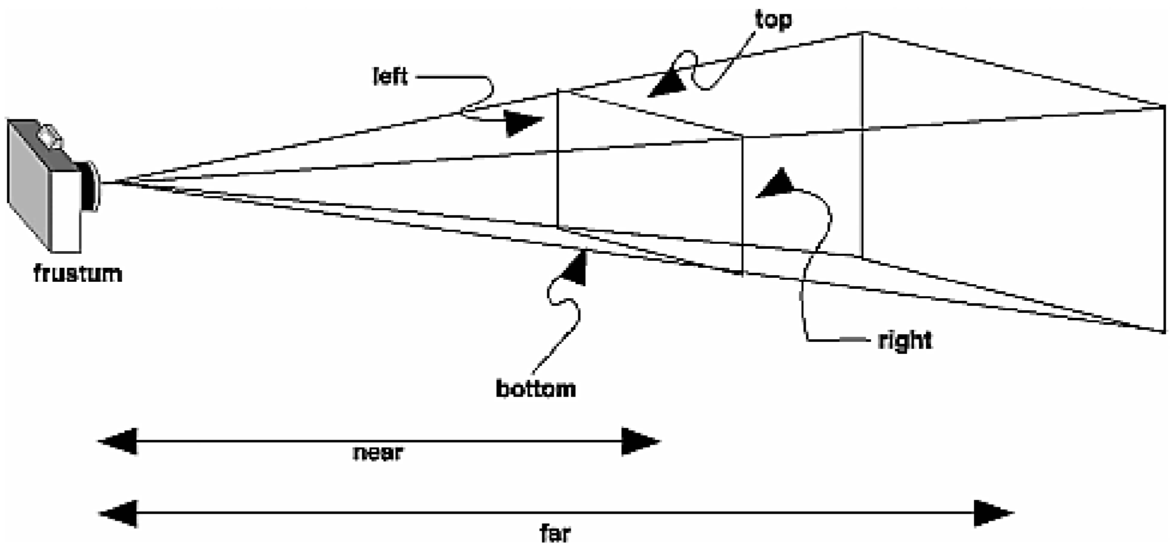
```
glRotatef(d,rx,ry,rz);
glTranslate3f(tx,ty,tz);
```

```
glTranslate3f(tx,ty,tz);
glRotatef(d,rx,ry,rz);
```

Modelview Transformations

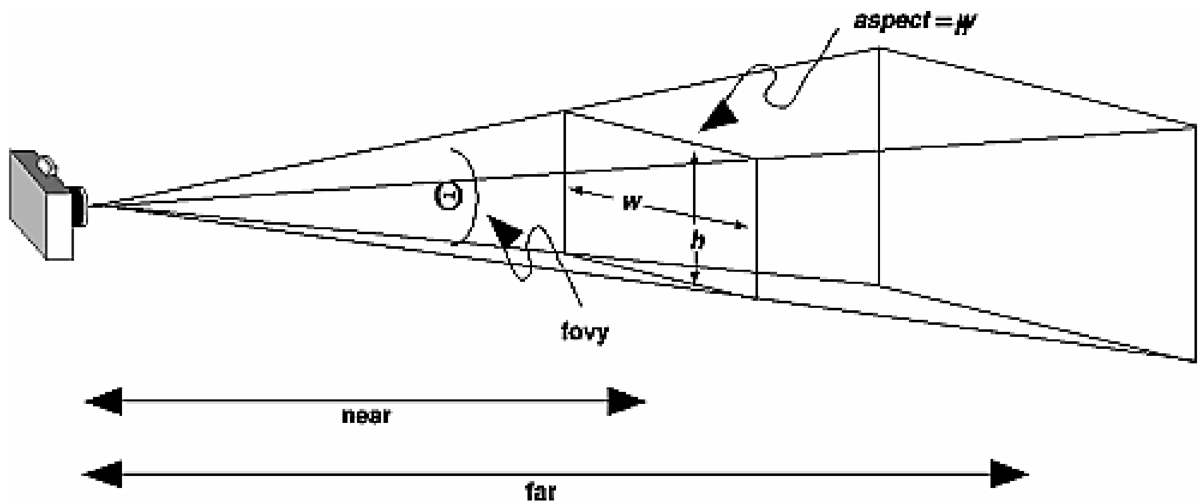
```
void pilotView{GLdouble planex, GLdouble planey, GLdouble
   planez, GLdouble roll, GLdouble pitch, GLdouble heading)
{
    glRotated(roll, 0.0, 0.0, 1.0);
    glRotated(pitch, 0.0, 1.0, 0.0);
    glRotated(heading, 1.0, 0.0, 0.0);
    glTranslated(-planex, -planey, -planez);
}

void polarView{GLdouble distance, GLdouble twist, GLdouble
   elevation, GLdouble azimuth)
{
    glTranslated(0.0, 0.0, -distance);
    glRotated(-twist, 0.0, 0.0, 1.0);
    glRotated(-elevation, 1.0, 0.0, 0.0);
    glRotated(azimuth, 0.0, 0.0, 1.0);
}
```

Projection Transformations



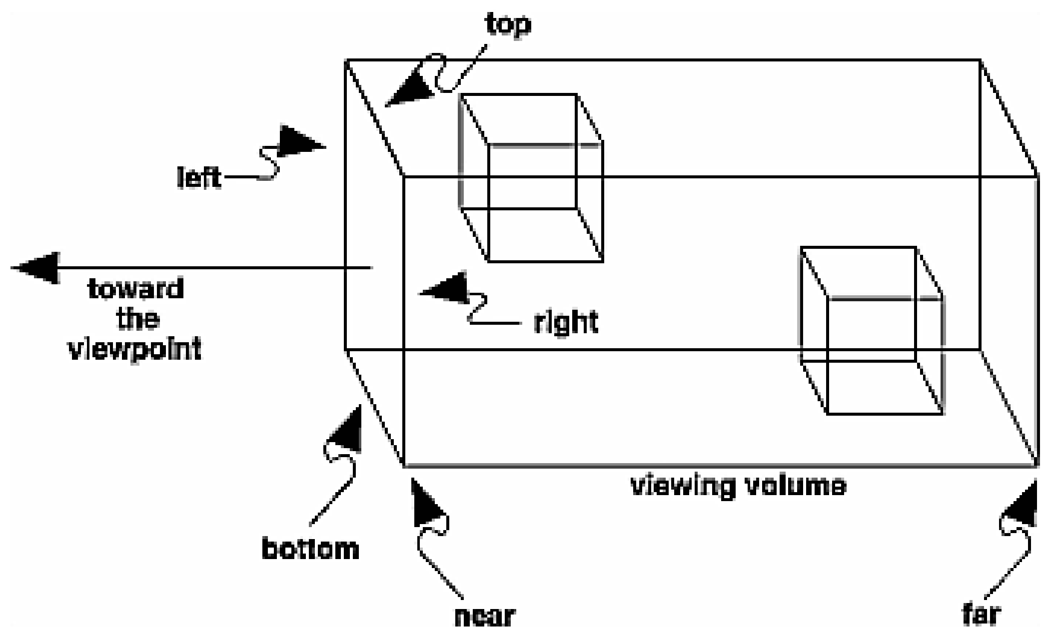

```
void glFrustum(GLdouble left, GLdouble right, GLdouble
  bottom, GLdouble top, GLdouble near, GLdouble far);
```

Projection Transformations




```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble
   near, GLdouble far);
```

Projection Transformations




```
void glOrtho(GLdouble left, GLdouble right, GLdouble
  bottom,
  GLdouble top, GLdouble near, GLdouble far);

void gluOrtho2D(GLdouble left, GLdouble right,
  GLdouble bottom, GLdouble top);
```

Matrix Transformations

```
draw_wheel_and_bolts()
{
   long i;
   draw_wheel();
   for(i=0;i<5;i++)
   {
       glPushMatrix();
       glRotatef(72.0*i,0.0,0.0,1.0);
       glTranslatef(3.0,0.0,0.0);
       draw_bolt();
       glPopMatrix();
   }
}
```

Simple OpenGL Program

```
{
   <Initialize OpenGL state>

   <Load and define textures>

   <Specify lights and shading parameters>

   <Load projection matrix>

   For each frame

        <Load model view matrix>
        <Draw primitives>

   End frame
}
```

Simple Program

```
#include <GL/gl.h>
main()
{
   InitializeAWindowPlease();
   glMatrixMode(GL_PROJECTION);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslate3f(1.0, 1.0, 1.0):
   glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
   glEnd();
   glFlush();
   UpdateTheWindowAndCheckForEvents();
}
```

GLUT

- = Graphics Library Utility Toolkit
  - Adds functionality such as windowing operations to OpenGL
- Event-based callback interface
  - Display callback
  - Resize callback
  - Idle callback
  - Keyboard callback
  - Mouse movement callback
  - Mouse button callback

Simple OpenGL + GLUT Program

```
#include <…>

DisplayCallback()
{
   <Clear window>
   <Load Projection matrix>
   <Load Modelview matrix>
   <Draw primitives>
   (<Swap buffers>)
}

IdleCallback()
{
   <Do some computations>
   <Maybe force a window refresh>
}

KeyCallback()
{
   <Handle key presses>
}
```

```
KeyCallback()
{
   <Handle key presses>
}

MouseMovementCallback
{
   <Handle mouse movement>
}

MouseButtonsCallback
{
   <Handle mouse buttons>
}

Main()
{
   <Initialize GLUT and callbacks>
   <Create a window>
   <Initialize OpenGL state>

   <Enter main event loop>
}
```

Simple OpenGL + GLUT Program

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);
  glLoadIdentity ();
  gluLookAt (0, 0, 5, 0, 0, 0, 0, 1, 0);
  glScalef (1.0, 2.0, 1.0);
  glutWireCube (1.0);
  glFlush ();
}
```

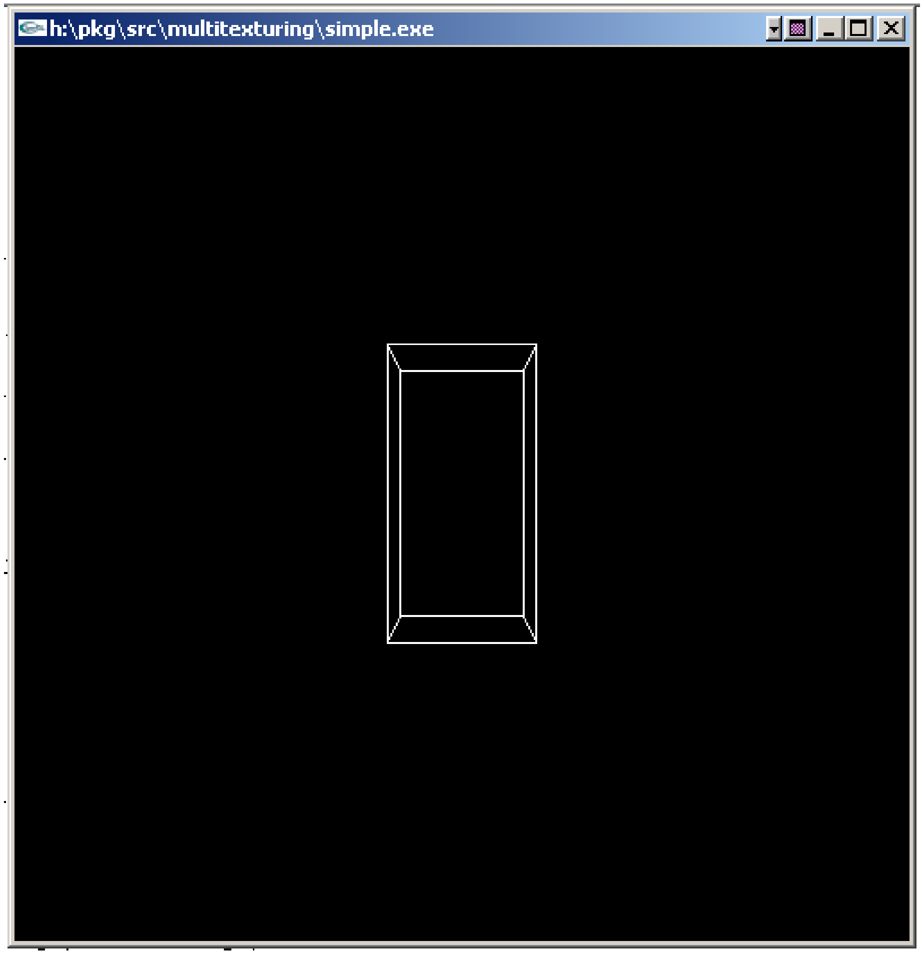
```
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
  glMatrixMode (GL_MODELVIEW);
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

Example Program with Lighting

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

void init(void)
{
  GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat mat_shininess[] = { 50.0 };
  GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_SMOOTH);

  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
}
void display(void)
{
  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glutSolidSphere (1.0, 20, 16);
  glFlush ();
}
```

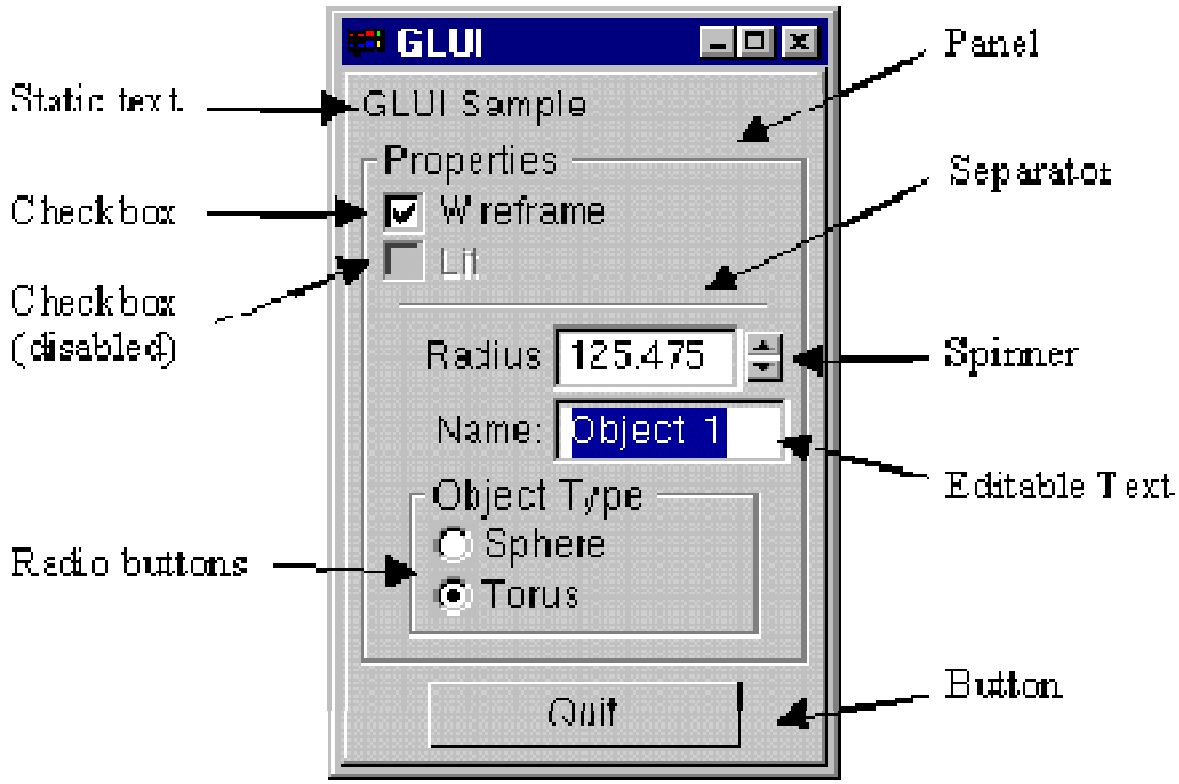
```
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
    glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
      1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
  else
    glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
      1.5*(GLfloat)w/(GLfloat)h, -1.5, 1.5, -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB |
        GLUT_DEPTH);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

Simple OpenGL + GLUT Program



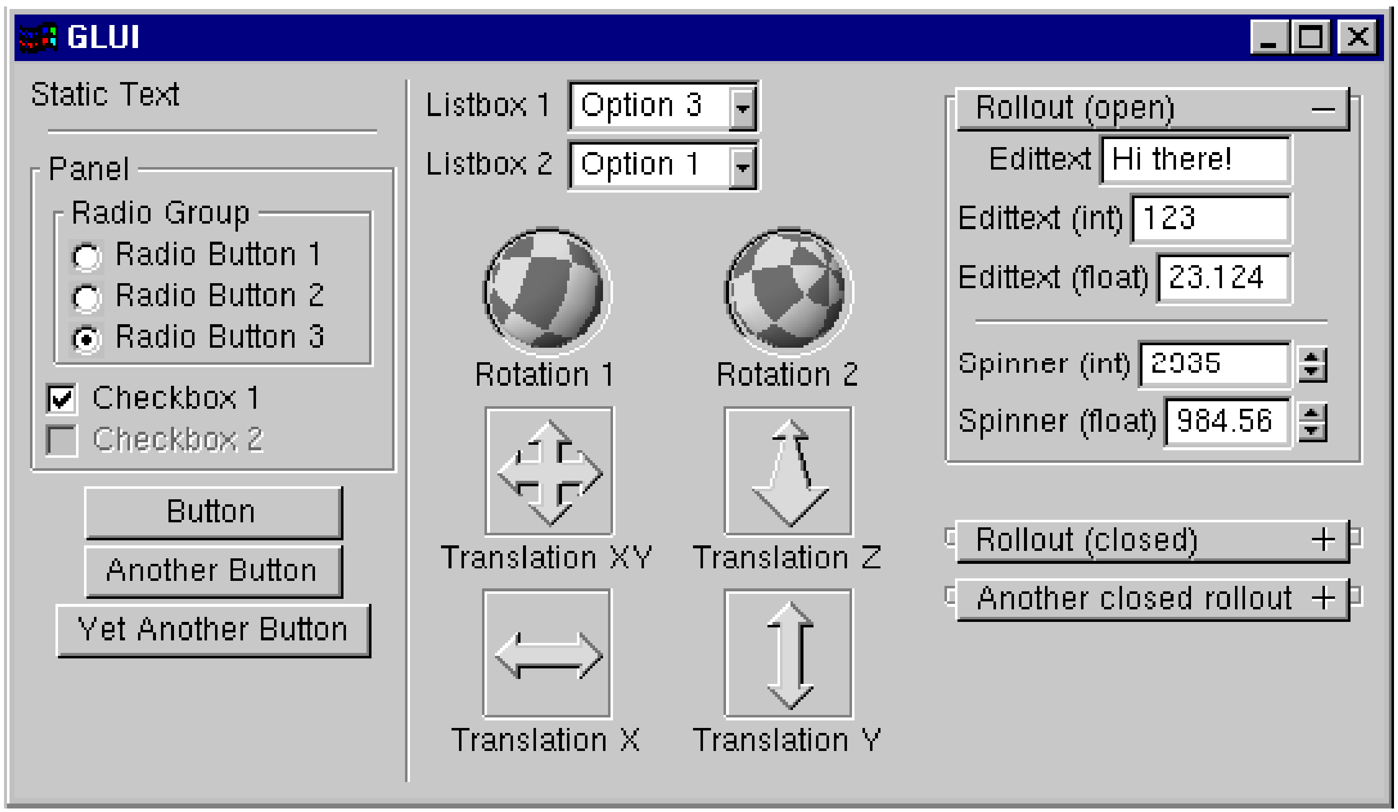
GLUI

- = Graphics Library User Interface



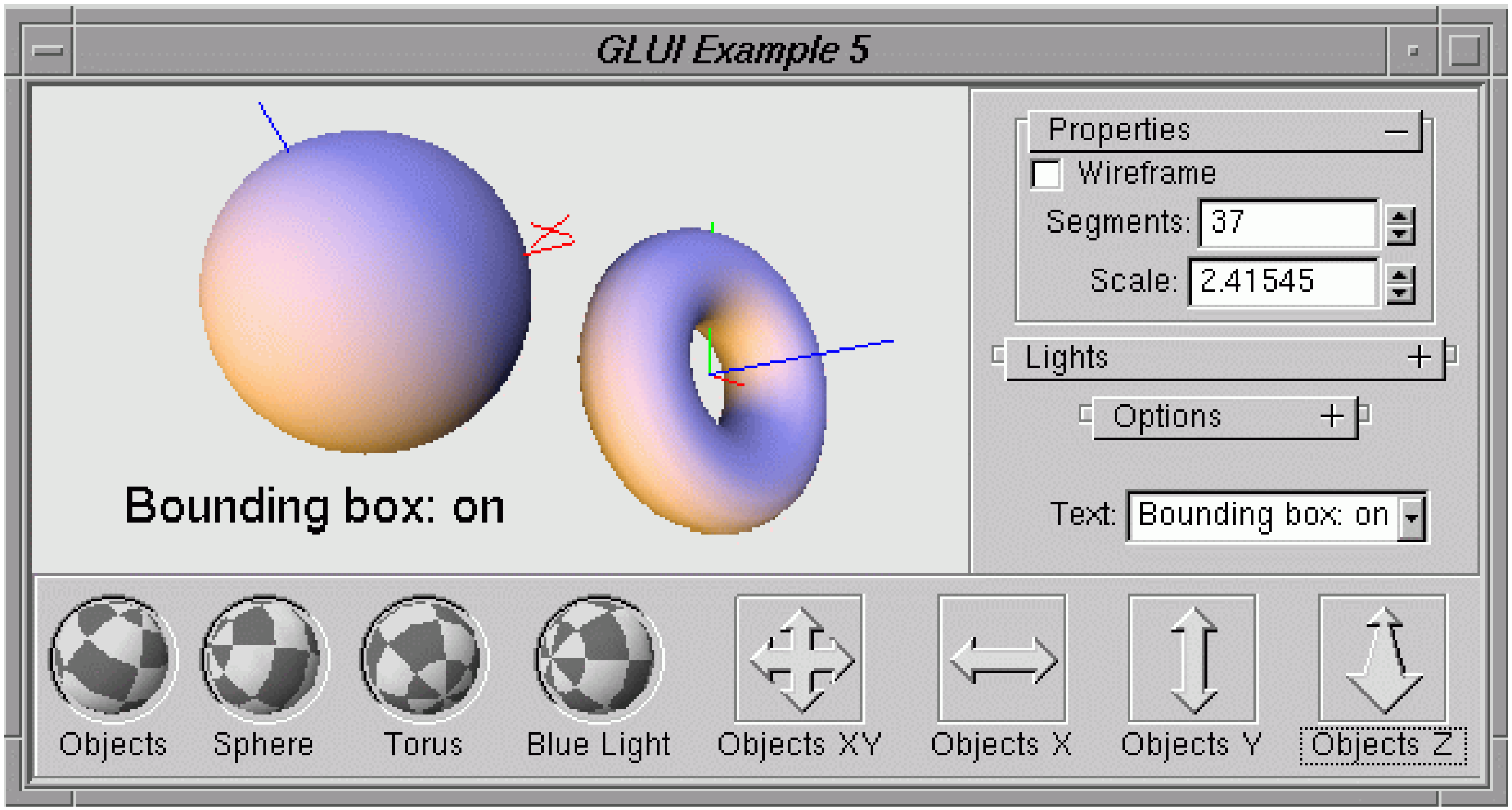
GLUI

- = Graphics Library User Interface



GLUI

- = Graphics Library User Interface



Alternatives graphics pipeline?

- Traditional pipeline…ok
- Parallel pipeline
  - Cluster of PCs?
  - Cluster of PS3?
  - What must be coordinated? What changes? What are the bottlenecks?

  - Sort-first vs. Sort-last pipeline
    - PixelFlow
    - Several hybrid designs

What can you do with a graphics pipeline?

- Uhm…graphics

What can you do with a graphics pipeline?

- Uhm…graphics
- Paperweight?

What can you do with a graphics pipeline?

- Uhm…graphics
- Paperweight?

- How about large number crunching tasks?
- How about general (parallelizable) tasks?

CUDA and OpenCL

- NVIDIA defined "CUDA" (new)
  – Compute Unified Device Architecture
  – http://www.nvidia.com/object/cuda_home.html#

- Khrono's group defined "OpenCL" (newer)
  – Open Standard for Parallel Programming of Heterogeneous Systems
  – http://www.khronos.org/opencl/

CUDA Example

- Rotate a 2D image by an angle
  - On the CPU (PC)
    - simple-tex.pdf
  - On the GPU (graphics card)
    - simple-tex-kernel.pdf

OpenCL Example

- Compute a Fast Fourier Transform
  - On the CPU (PC)
    - cl-cpu.pdf
  - On the GPU (graphics card)
    - cl-gpu.pdf

OpenCV

- A library for computer-vision related software
- Derived from research work and high-performance code from Intel
- http://opencv.willowgarage.com/wiki/
  - e.g., find fundamental matrix