CS535
Interactive Computer Graphics

Fall 2024

Daniel G. Aliaga
Department of Computer Science
Purdue University

Who am I?

- Daniel G. Aliaga

  http://www.cs.purdue.edu/~aliaga and aliaga@cs.purdue.edu

  CS faculty doing Graphics

  Doctorate in Graphics

  Master's in Graphics

  Bachelors in Graphics

  High School Degree doing graphics/robots/science

  1980 (TRS80 Model I)

  Then: http://www.youtube.com/watch?v=3yuqdC8Id48)

  http://thinkingscifi.files.wordpress.com/2012/12/starwars-graphics.png

  Now: http://www.youtube.com/watch?v=QAEkuVgt6Aw

- CGVLAB

  http://www.cs.purdue.edu/cgvlab

Who am I?

- CGVLAB: www.cs.purdue.edu/cgvlab
- Home page: www.cs.purdue.edu/homes/aliaga

- Research Computer Graphics/Computer Vision:
  - Urban Modeling: 3D acquisition, forward and inverse procedural modeling, urban design and planning
  - Projector-Camera Systems: spatially-augmented reality, appearance editing, radiometric compensation
  - 3D digital fabrication: genuinity detection, tamper detection, multiple appearance generation

Who are you?

Syllabus

- History
- Graphics Pipeline
- Ray-tracing and Point Rendering
- Polygon Rendering
- Shading and Illumination

(midterm)

- Image-based Rendering
- Generating Modeling
- Style and Appearance

(final project, final exam)

Preview: CS635

- Neural Networks, CNNs, GANs
- More 3D Deep Learning
- Surface Reconstruction
- Probabilistic Graphical Models
- 3D Reconstruction Passive and Active
- Fancy Cameras and Displays
- Perception Issues
- Generative Modeling

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

CS535 Fall 2024

Daniel G. Aliaga
Department of Computer Science
Purdue University

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

Reprise: Graphics

- First graphics **visual** image:
  - Ben Laposky used an oscilloscope in 1950s

(note: one of my undergrad senior projects was an oscilloscope based graphics engine)

Whirlwind Computer @ MIT

- Video display of real-time data:

Ivan Sutherland (1963) - SKETCHPAD

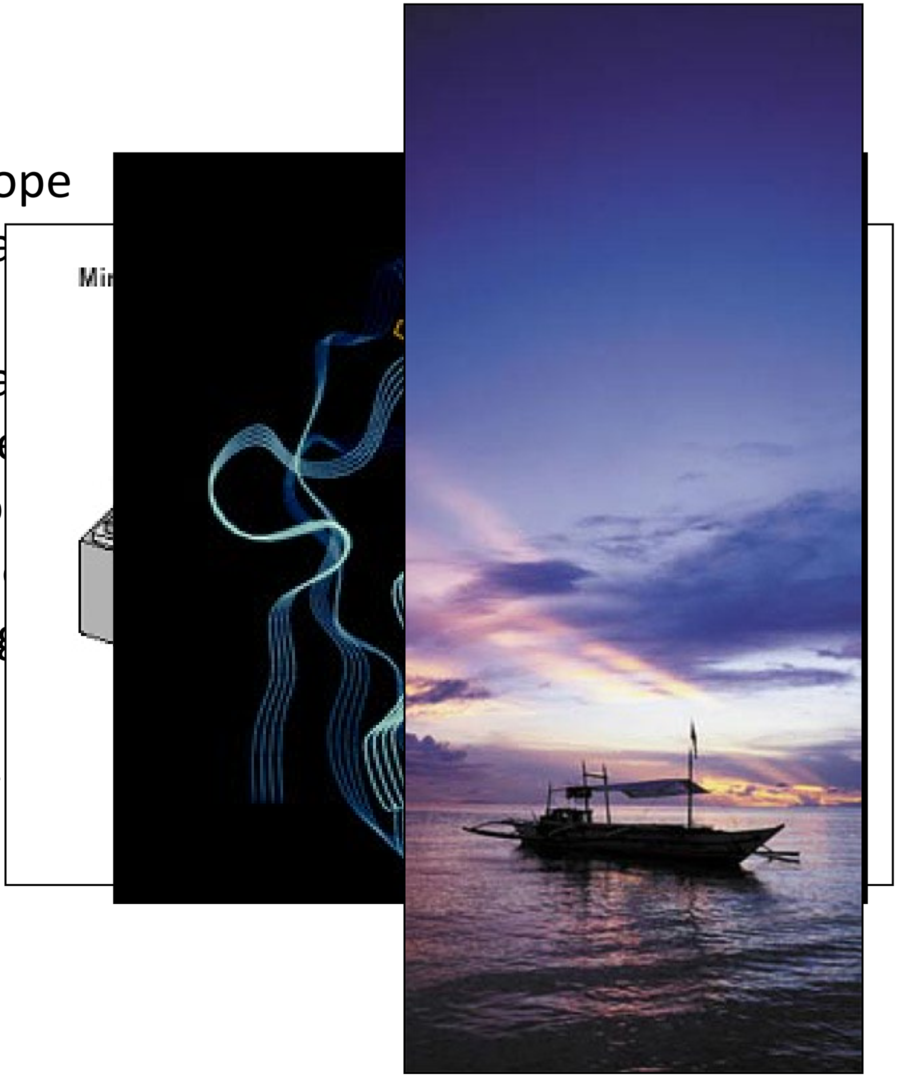
- pop-up menus
- constraint-based drawing
- hierarchical modeling

IKONAS and TAAC

- Nick England and more…
- (see other slides)

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
  -
  -

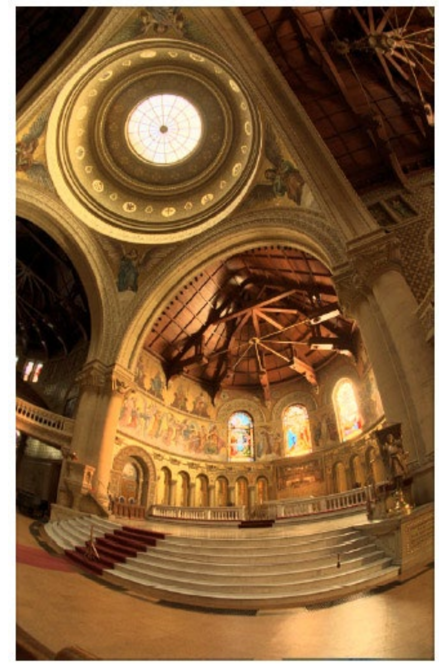
Input hardware

- 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

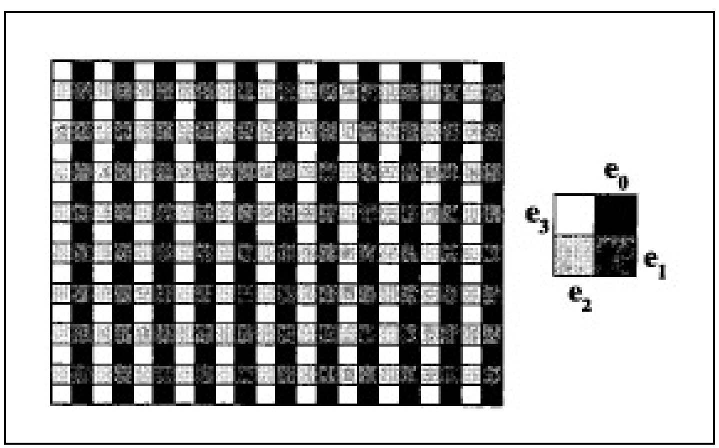
High Dynamic Range Imaging

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

→



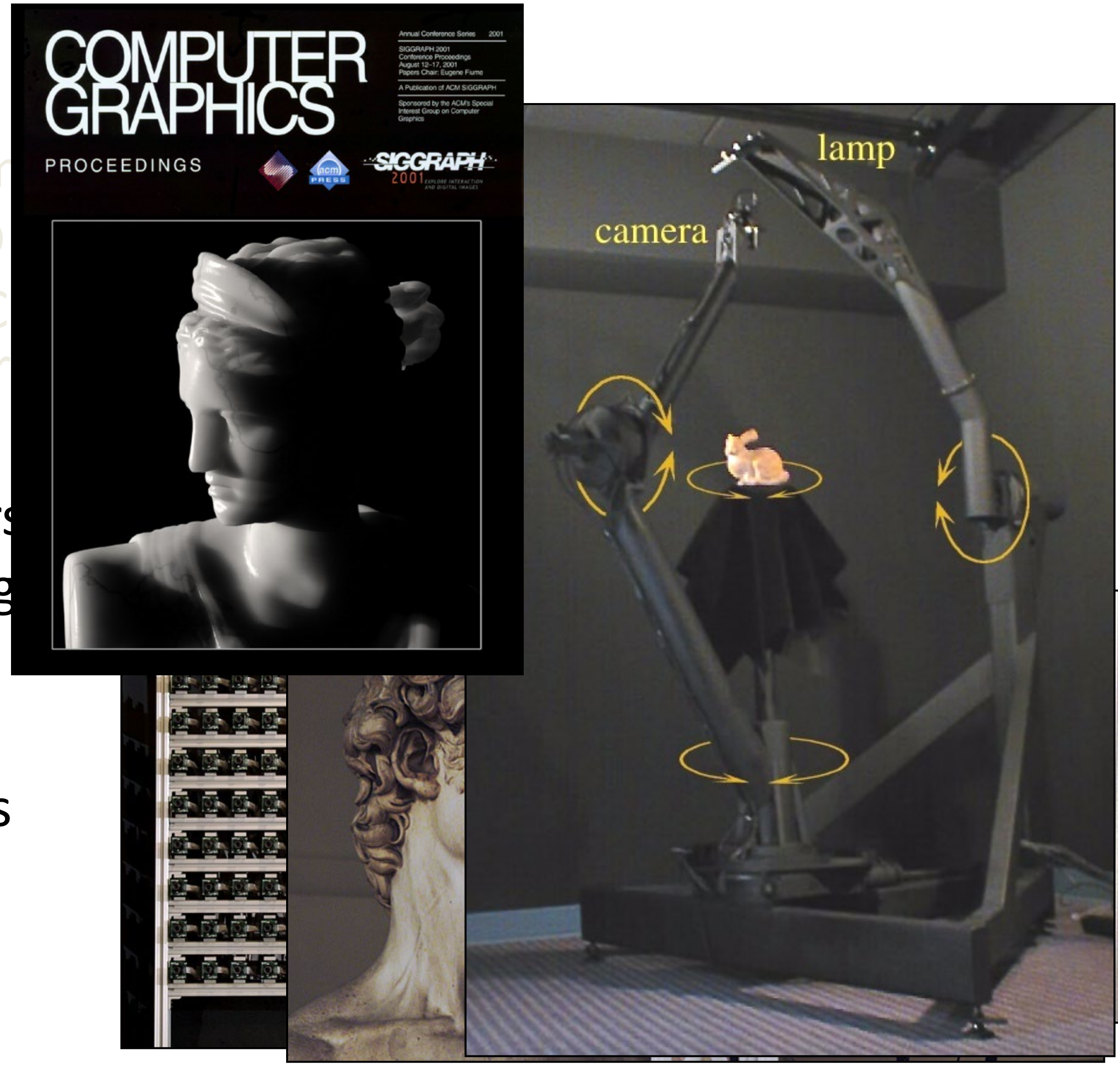
[Debevec97]



[Nayar00]

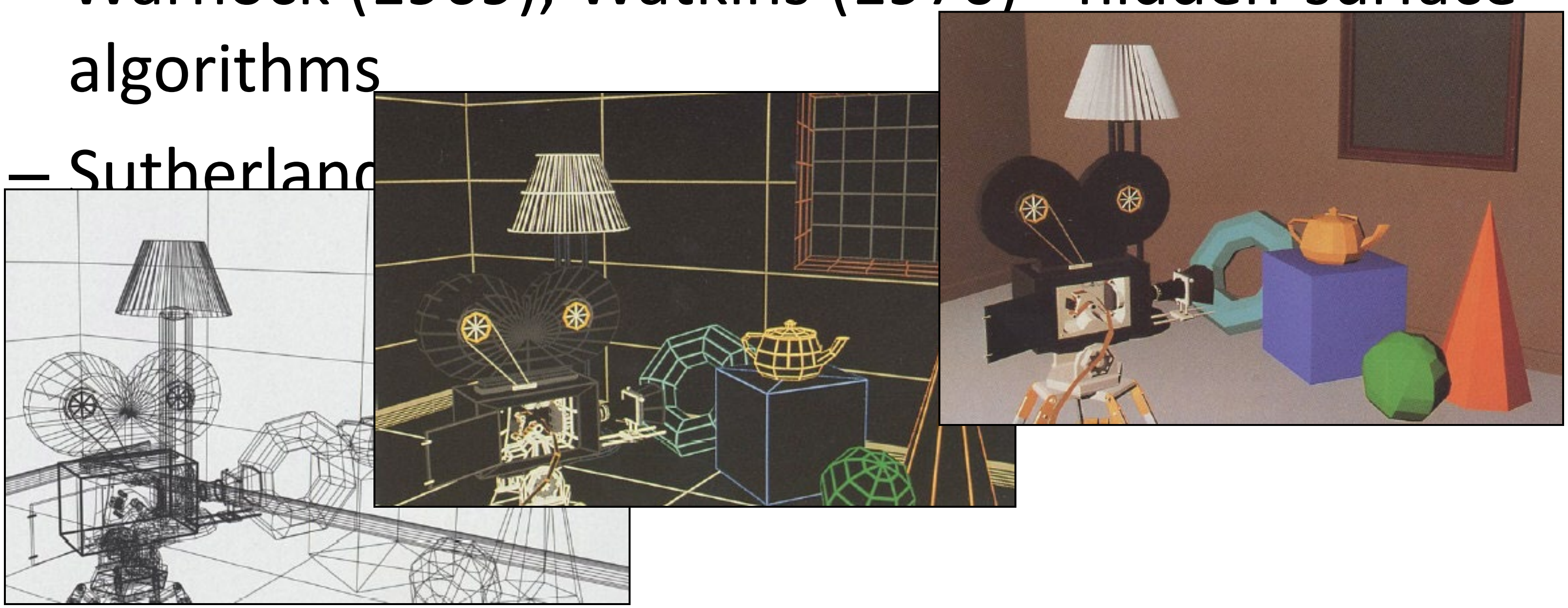
Input hardware

- 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

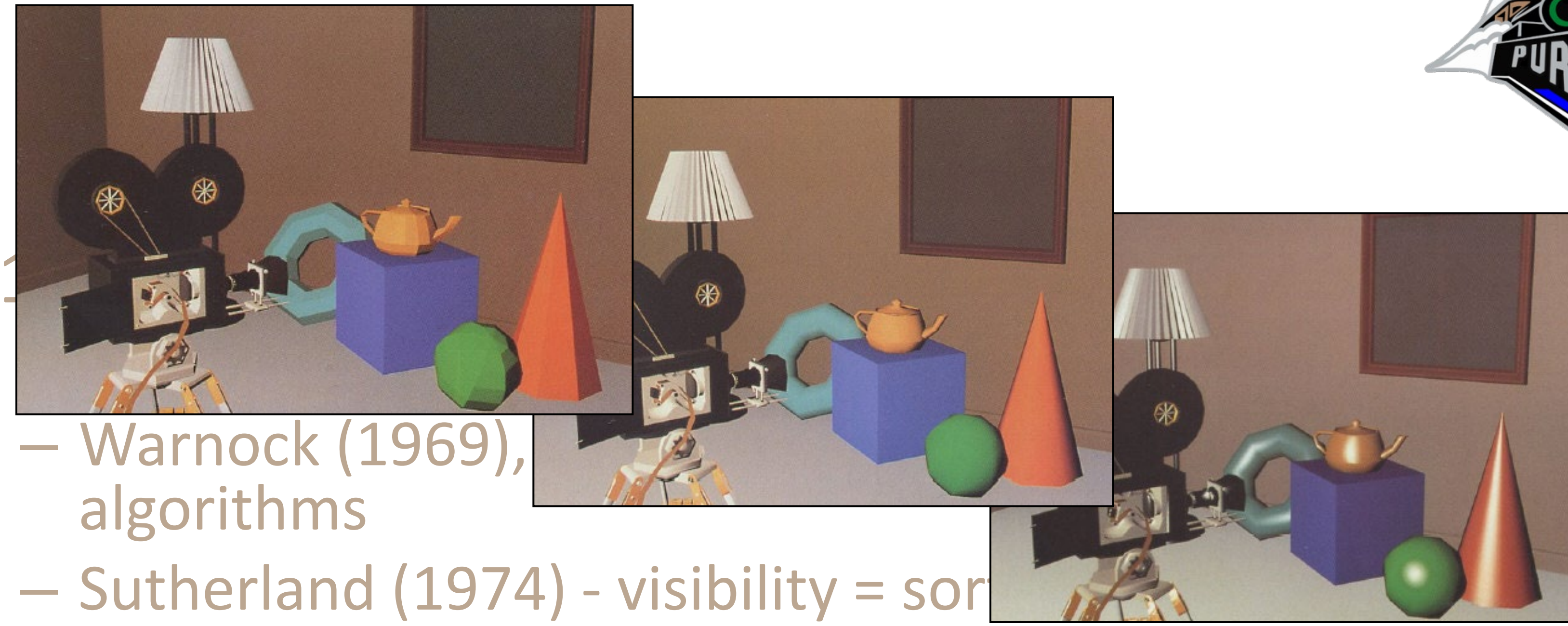


Rendering

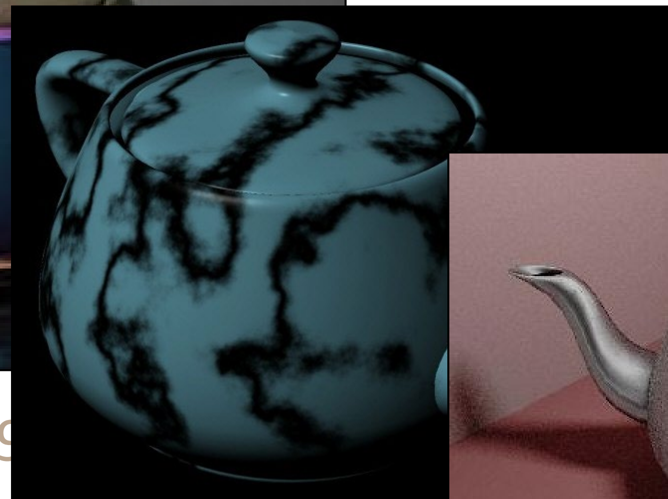
- 1960s - the visibility problem
  - Roberts (1963), Appel (1967) - hidden-line algorithms
  - Warnock (1969), Watkins (1970) - hidden-surface algorithms
  - Sutherland



- 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



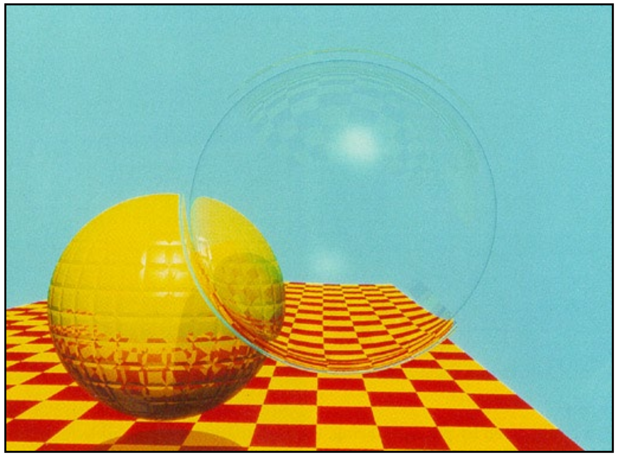
- 1

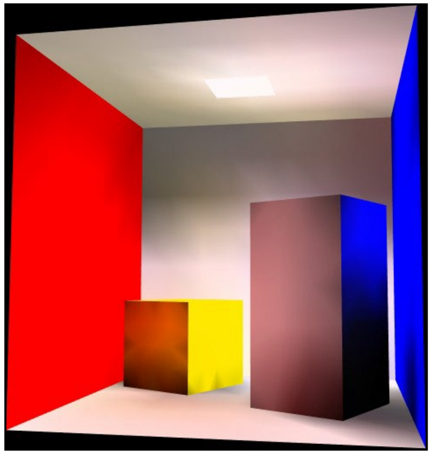
  algorithms

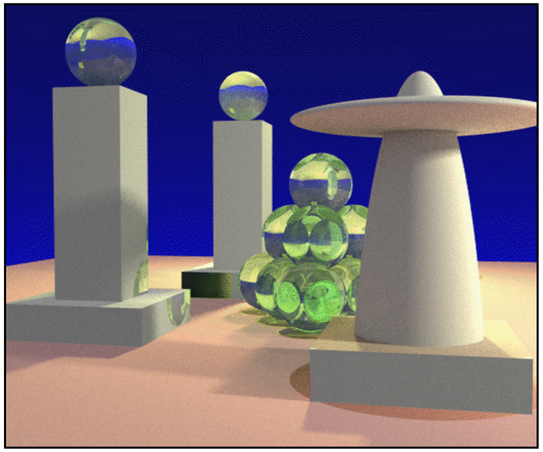
  – Sutherland (19

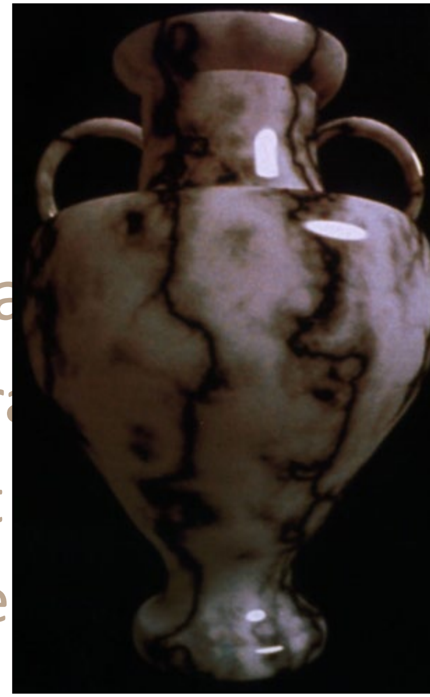
- 1970s - raster graphics
  - Gouraud (1971) - diffuse lighting
  - Phong (1974) - specular lighting
  - Blinn (1974) - curved surfaces, texture
  - Catmull (1974) - Z-buffer hidden-surface algorithm
  - Crow (1977) - anti-aliasing

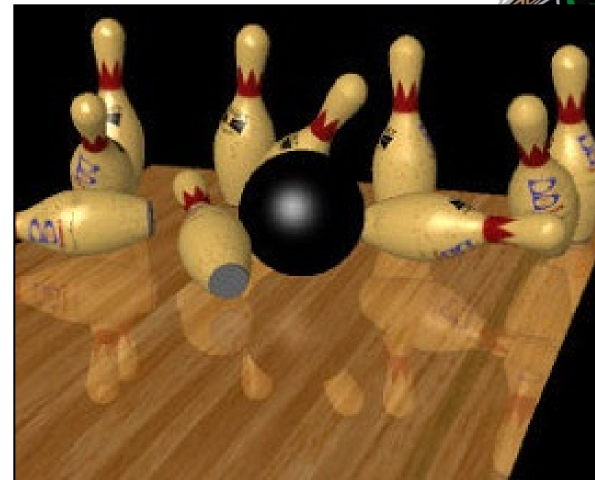
- early 1980s - global illumination
  - Whitted (1980) - ray tracing
  - Goral, Torrance et al. (1984), Cohen (1985) - radiosity
  - Kajiya (1986) - the rendering equation







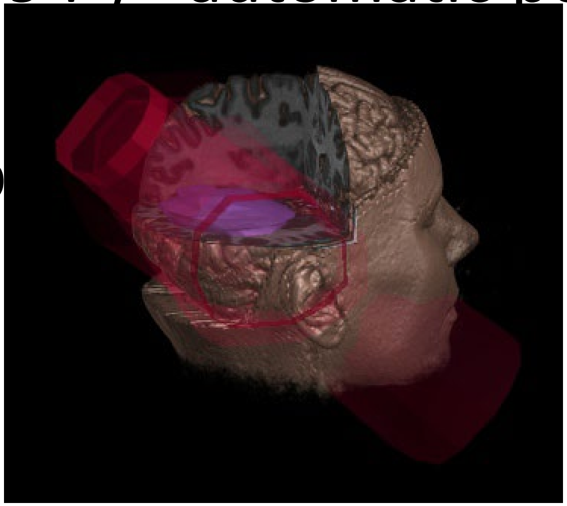
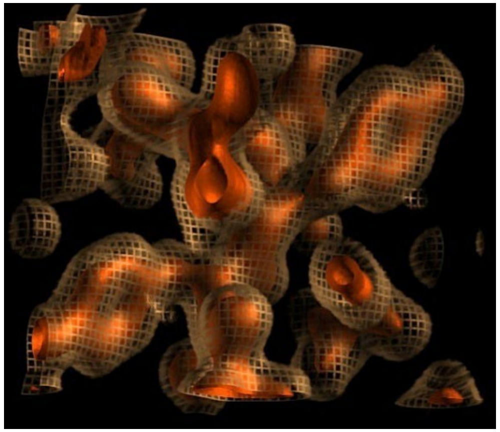




- e … pa … n
  - … r …
  - Goral, Torrance et … en (1985) - radiosity
  - Kajiya (1986) - the … ation

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

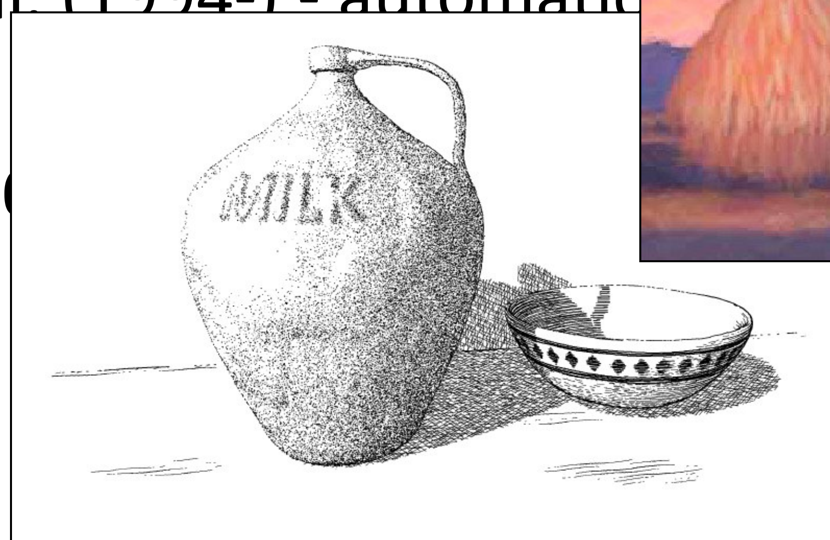
- early 1990s - non-photorealistic rendering
  - Drebin et al. (1988), Levoy (1988) - volume rendering
  - Haeberli (1990) - impressionistic paint programs
  - Salesin et al. (1994-) - automatic pen-and-ink

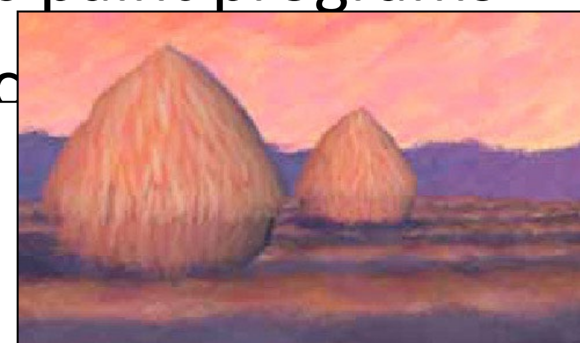




- early 1990s - non-photorealistic rendering
  - Drebin et al. (1988), Levoy (1988) - volume rendering
  - Haeberli (1990) - impressionistic paint programs
  - Salesin et al. (1994-) - automatic

n

9





Research Conferences…

- Papers at http://kesen.realtimerendering.com/

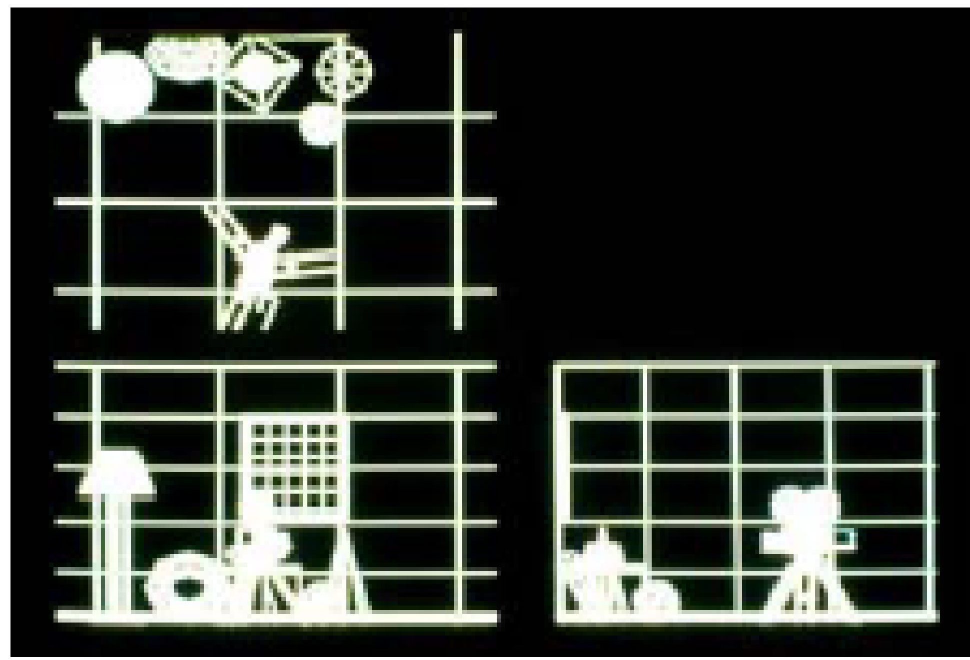
- SIGGRAPH, SIGGRAPH Asia, Eurographics, I3D
- CVPR, ICCV, ECCV…
- NeurIPS, AAAI, ICLR, ICML…
- IEEE Visualization

Computer Graphics Pipeline

- How do we create a rendering such as this?

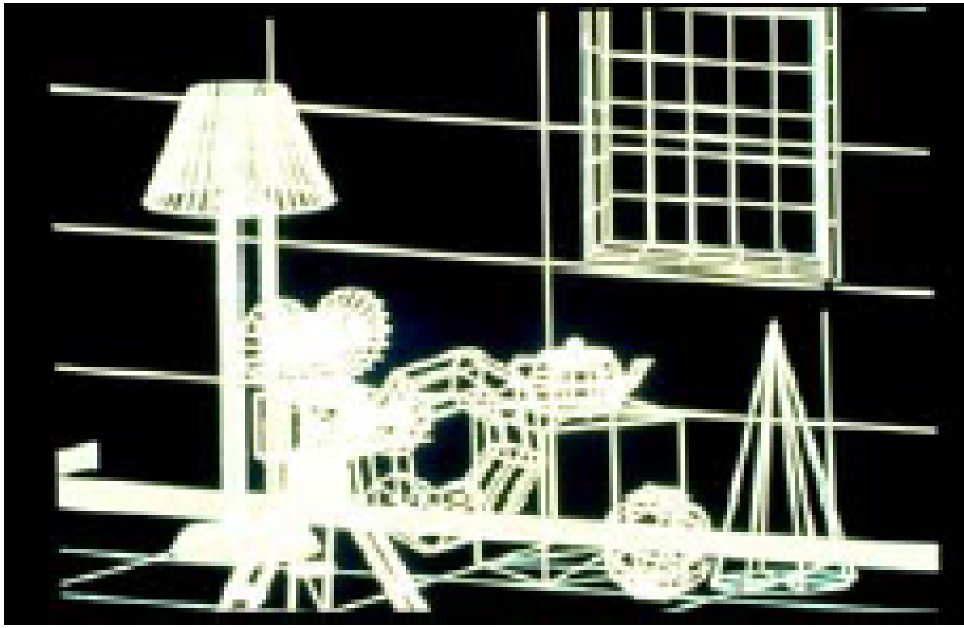
Computer Graphics Pipeline

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



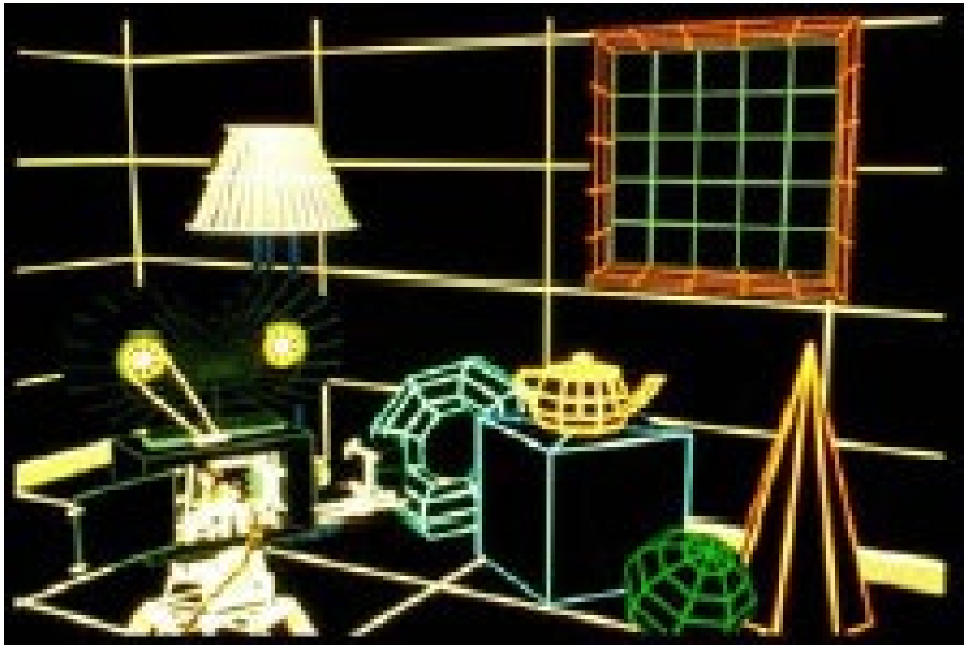
Computer Graphics Pipeline

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



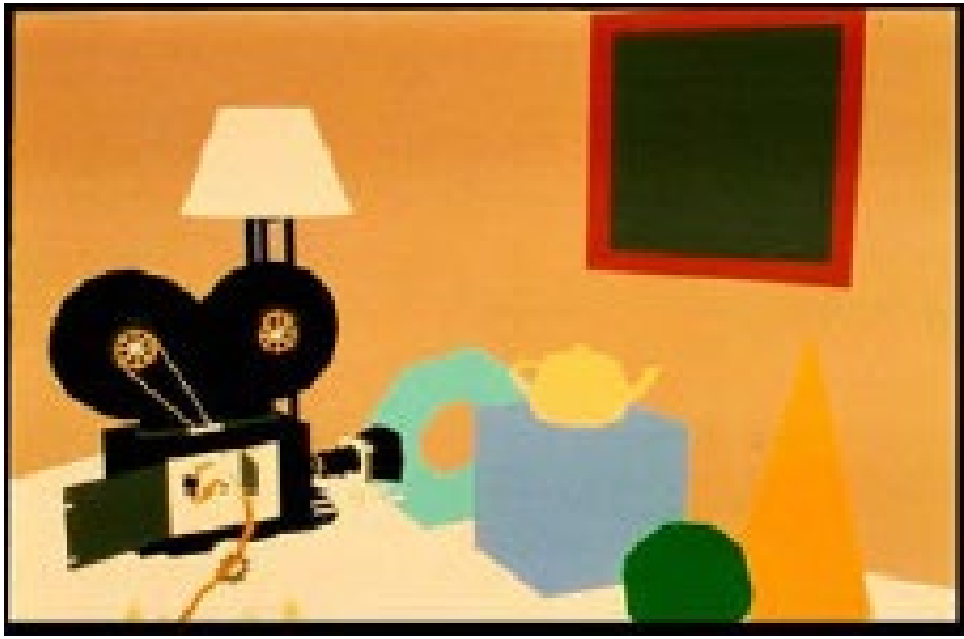
Computer Graphics Pipeline

- 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

But…

- Now, we have deep learning…

- Or, did we always?

Deep Visual Computing

- Since the beginning, it turns out **visual computing** and **machine learning** have been **deeply** connected

- Do you know why?

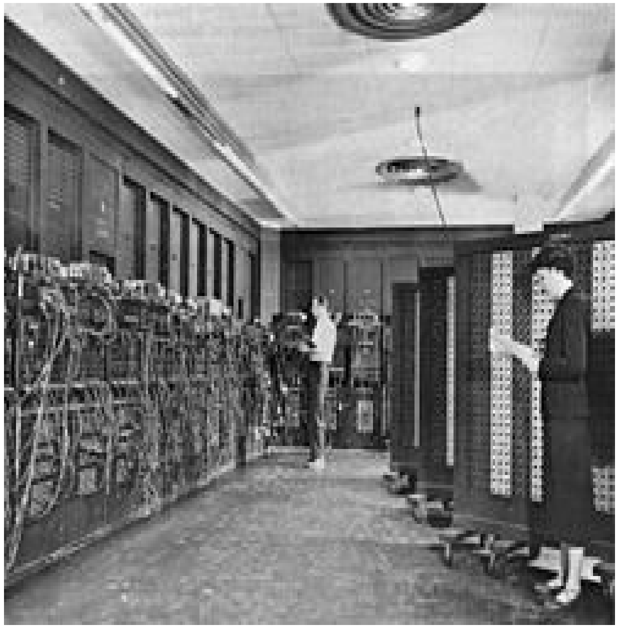
- Lets see… (get it: lets “see”)

A long time ago in a computer far, far inferior to your phone, it all began…

-Daniel Aliaga, August 25, 2020

ENIAC

- Completed in 1945
- Was called a “Giant Brain”
- Cost $6.3M of today’s dollars



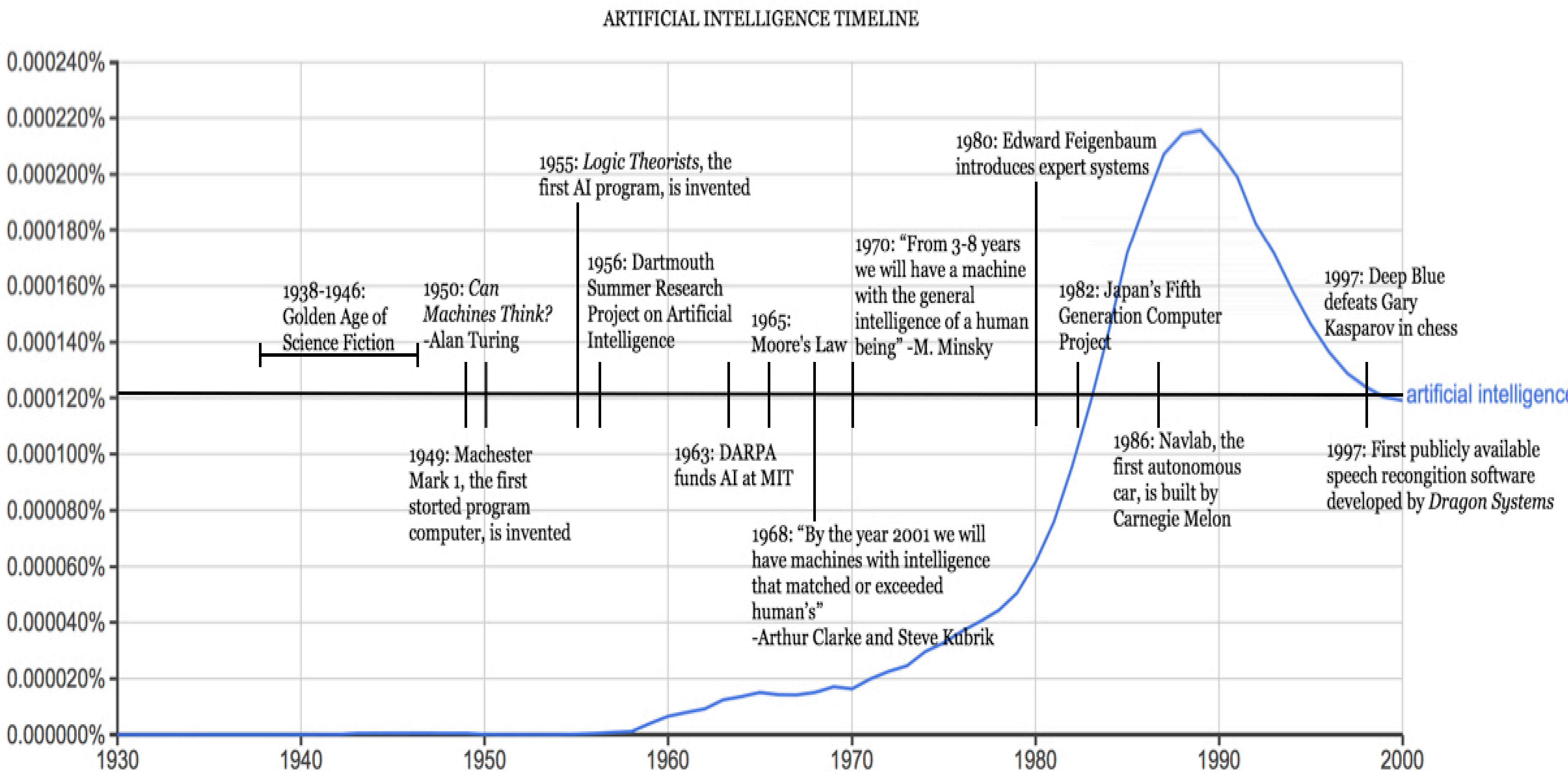
- However, computers then lacked a key prerequisite for intelligence:

  they could barely remember…they only executed a few commands

Logic Theorist (1956)

- A program designed to mimic the problem solving skills of a human
- From 1957-1974, AI flourished and failed and flourished…
- In 1968, A. Clarke and S. Kubrik said “by the year 2001 we will have machines with intelligence that matches or exceeded humans’s”
- In 1970, Marvin Minsky (MIT) said that in 3-8 years “we will have a machine with the general intelligence of an average human being”

AI Timeline


ARTIFICIAL INTELLIGENCE TIMELINE
0.000240%
0.000220%
0.000200%
0.000180%
0.000160%
0.000140%
0.000120%
0.000100%
0.000080%
0.000060%
0.000040%
0.000020%
0.000000%
1930
1940
1950
1960
1970
1980
1990
2000
artificial intelligence
1938-1946: Golden Age of Science Fiction
1950: *Can Machines Think?* -Alan Turing
1955: *Logic Theorists*, the first AI program, is invented
1956: Dartmouth Summer Research Project on Artificial Intelligence
1965: Moore's Law
1970: "From 3-8 years we will have a machine with the general intelligence of a human being" -M. Minsky
1980: Edward Feigenbaum introduces expert systems
1982: Japan's Fifth Generation Computer Project
1997: Deep Blue defeats Gary Kasparov in chess
1949: Machester Mark 1, the first storted program computer, is invented
1963: DARPA funds AI at MIT
1968: "By the year 2001 we will have machines with intelligence that matched or exceeded human's" -Arthur Clarke and Steve Kubrik
1986: Navlab, the first autonomous car, is built by Carnegie Melon
1997: First publicly available speech recongition software developed by *Dragon Systems*

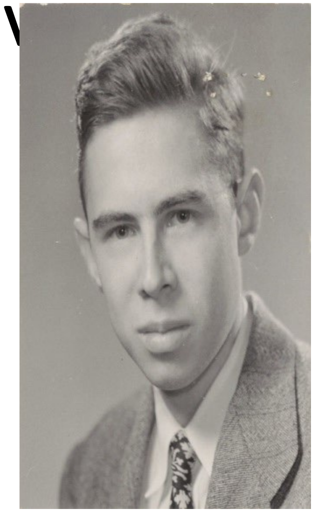
1980s

- Expert systems became popular: dedicated systems
- “Deep learning techniques” was a coined phrase but with diverse meanings…
- I was around then, and even a paid undergraduate researcher in a major AI lab

  - our job was to create a robot that could be programmed remotely and could execute algorithms for navigating and deciding how to avoid obstacles (e.g., walls and boxes)

(Single Layer) Perceptron

- The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, F. Rosenblatt, Psychological Rev
65(6), 1958.



- Model based on the human visual system

Perceptron

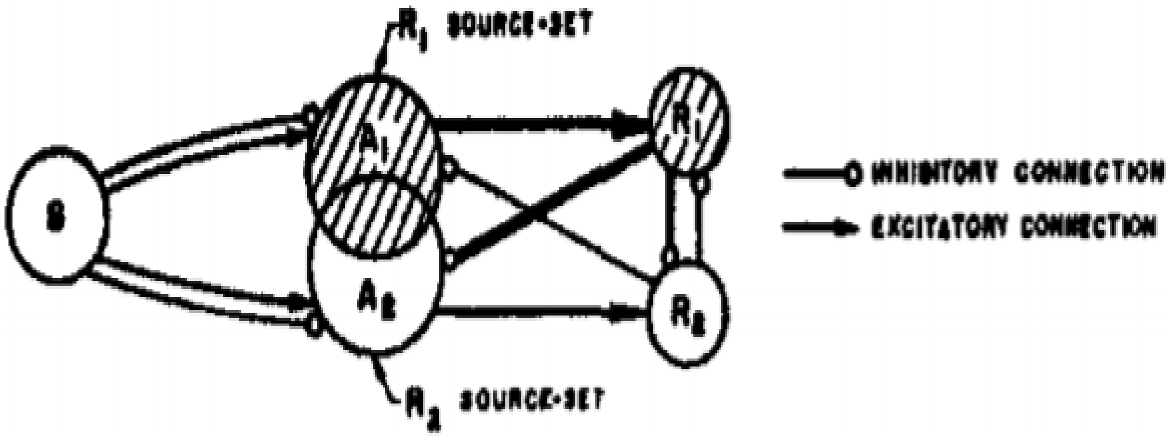


FIG. 2B. Venn diagram of the same perceptron (shading shows active sets for R_1 response).

Perceptron

Algorithm 1: Perceptron Learning Algorithm

Input: Training examples $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$.

Initialize $\mathbf{w}$ and b randomly.

while *not converged* **do**

  # # # Loop through the examples.

  for $j = 1, m$ **do**

    # # # Compare the true label and the prediction.

    $error = y_j - \sigma(\mathbf{w}^T \mathbf{x}_j + b)$

    ### If the model wrongly predicts the class, we update the weights and bias.

    if *error != 0* **then**

      ### Update the weights.

      $\mathbf{w} = \mathbf{w} + error \times x_j$

      ### Update the bias.

      $b = b + error$

  Test for convergence

Output: Set of weights $\mathbf{w}$ and bias b for the perceptron.

Perceptrons

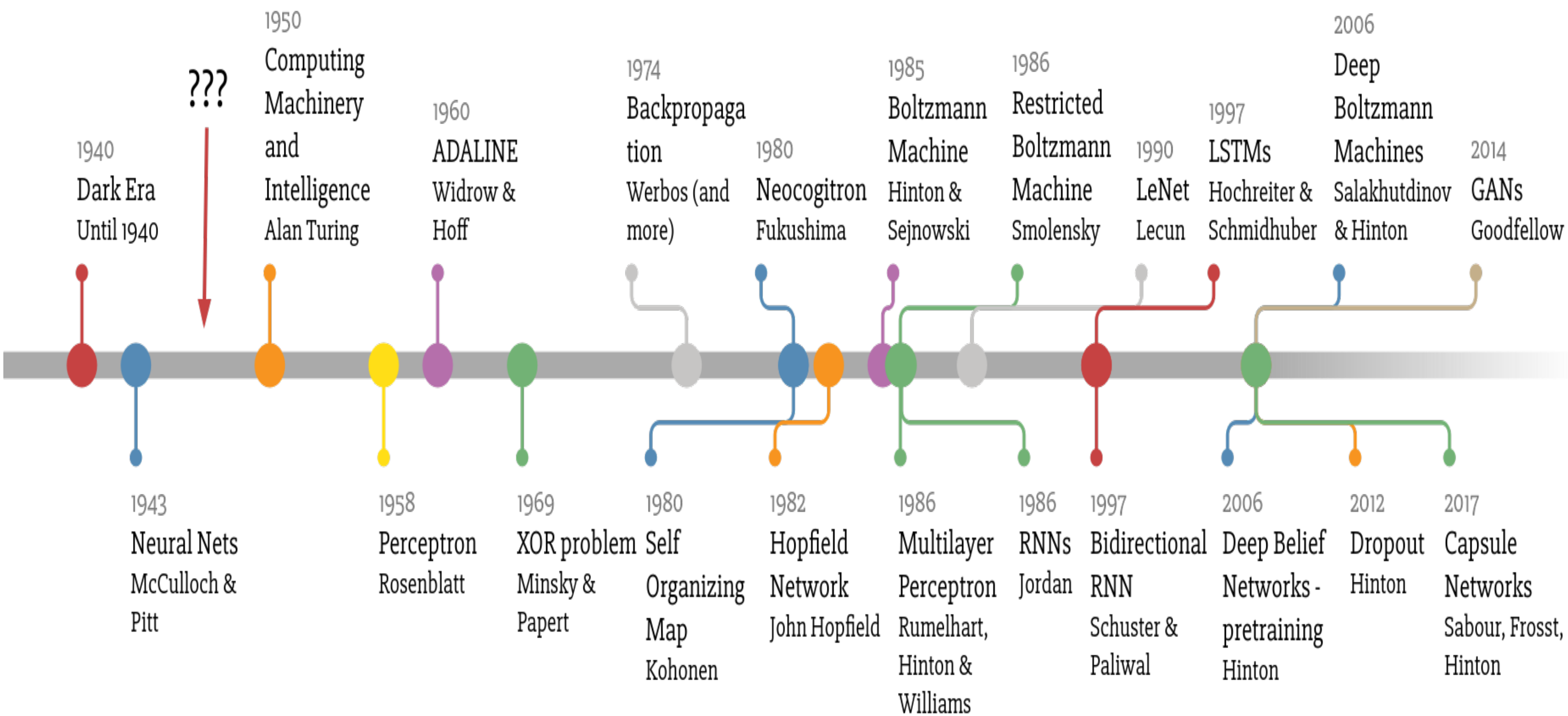
- Book by M. Minsky and S. Papert (1969)
- Was actually “An Introduction to Computational Geometry” – thus visual as well
- Commented on the limited ability of perceptrons and on the difficulty in training multi-layer perceptrons

Try this…

https://playground.tensorflow.org/

- First try something linear
- Then try something more complex…

Deep Learning Timeline



Made by Favio Vázquez

Reprise: Computer Vision

- In 1959, Russell Kirsch and colleagues developed an image scanner: transform an image into a grid of numbers so that a machine can understand it!

- One of the first scanned images:

  (176x176 pixels)

1982

- David Marr, British neuroscientists, published influential paper

  "Vision: A computational investigation into the human representation and processing of visual information"

  Among many things, he gave the insight that vision is hierarchical (i.e., primal sketch, 2.5D, and then 3D recognition)

  (now at CVPR, the Marr Prize exists)

1999

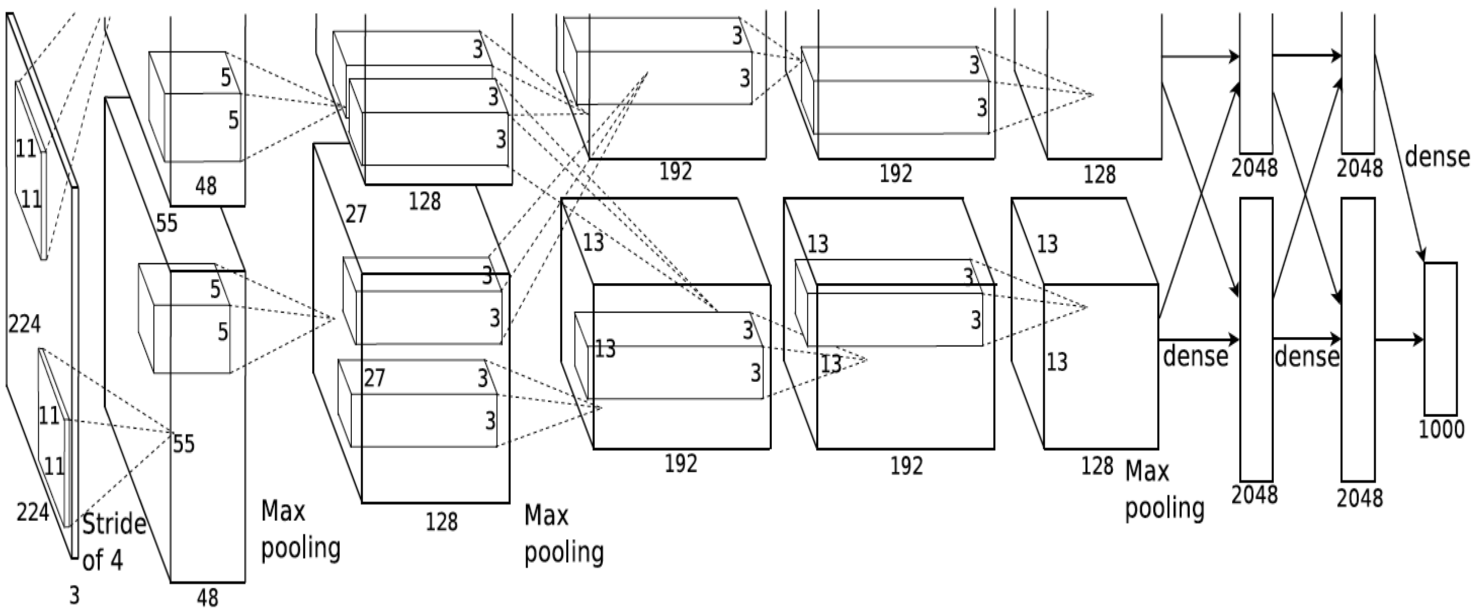
- David Lowe’s work “Object Recognition from Local Scale-Invariant Features” indicated a shift to feature-based visual object-recognition (instead of full 3D models as Marr proposed)
  - Scale-Invariant Feature Transform (SIFT)
  - and many subsequent derivatives

2010

- ImageNet Large Scale Visual Recognition Competition (ILSVRC) runs annually
  - 2010/2011: error rates were around 26% (using Lowe-style approaches)
  - 2012: the beginning of a new beginning – AlexNet – reduced errors to 16%!

AlexNet

- University of Toronto created a CNN model (AlexNet) that changed everything (Krizhevsky et al. 2012)



Just a note: 1980s

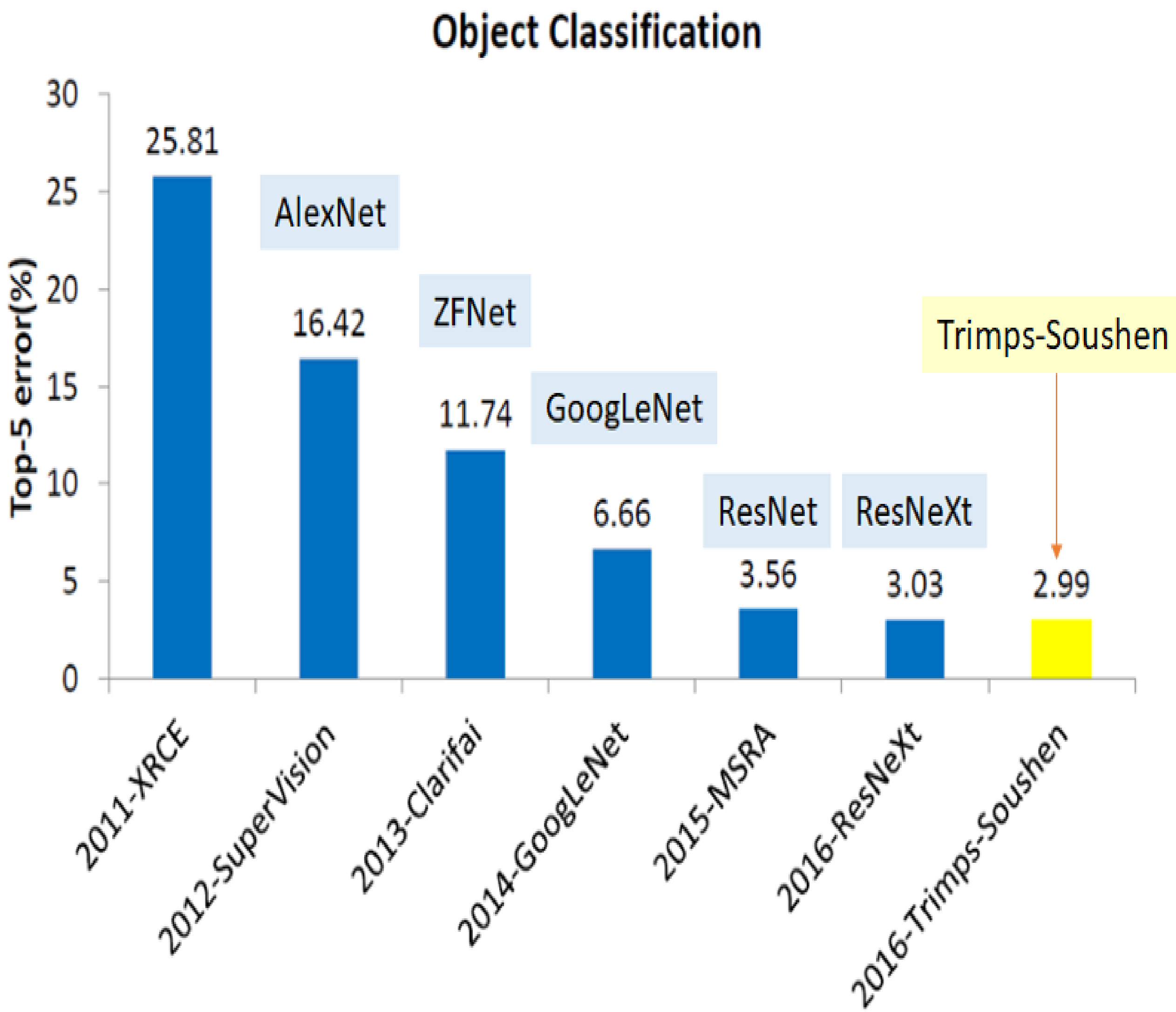
- Kunihiko Fukushima developed Neocognitron for visual pattern recognition which included several *convolutional* layers whose (typically rectangular) receptive fields had weight vectors (known as filters)

- This was perhaps the earliest deep and convolutional network

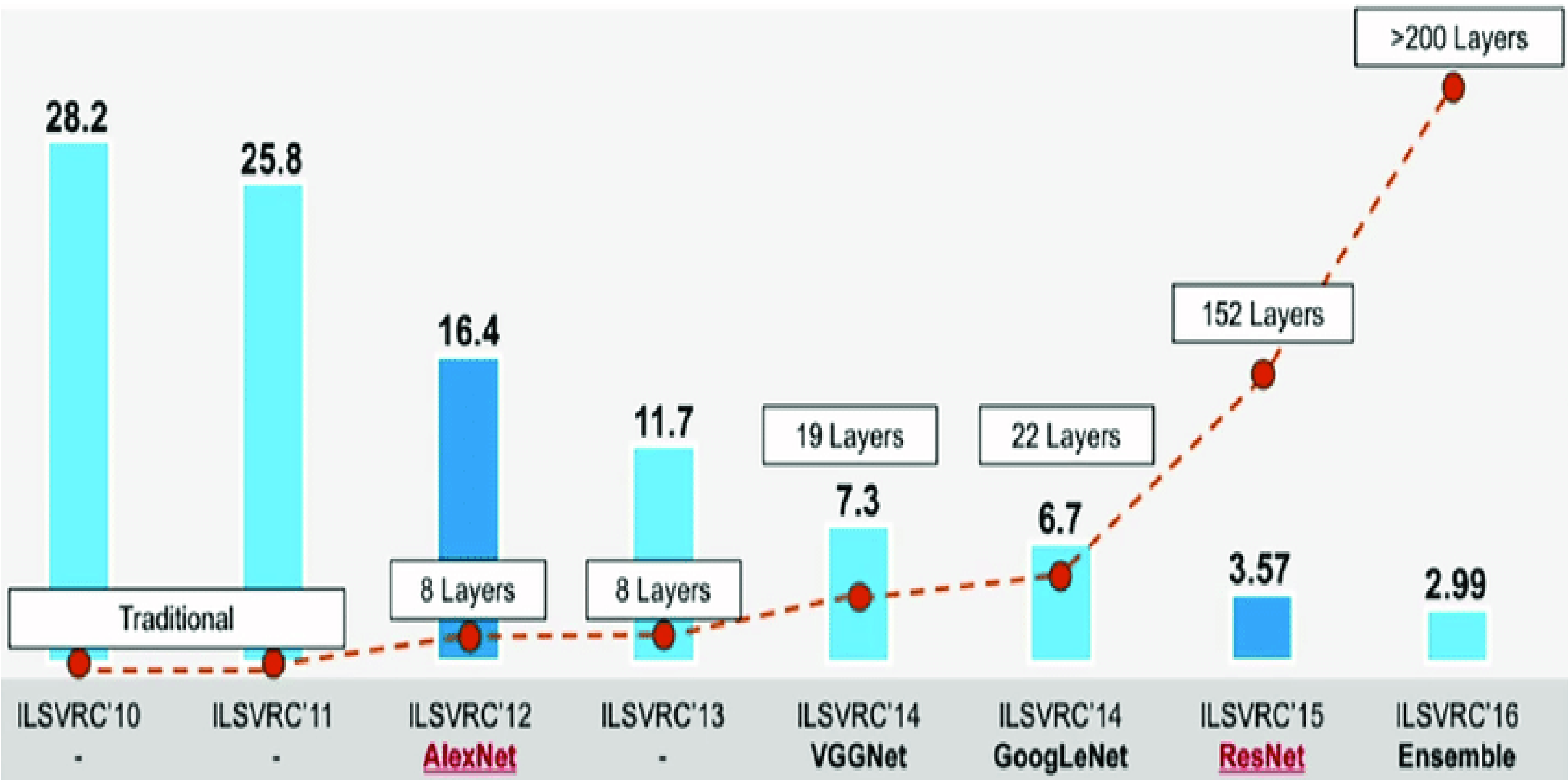
Just a note: 1989

- Yann LeCun applied backpropagation to Fukushima’s network and with other improvements released LeNet-5 – quite similar to today’s CNNs

ILSVRC (2011-2017)



ILSVRC (2010-2017)



Deep Learning in Computer Graphics

- Like in computer vision, since 2010'ish **deep learning** has revolutionized computational **imaging** and computational **photography**

- However, hand-crafted methods have significantly improved other domains such as **geometry processing, rendering and animation, video processing, and physical simulations**

Linear Algebra

- 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

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

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

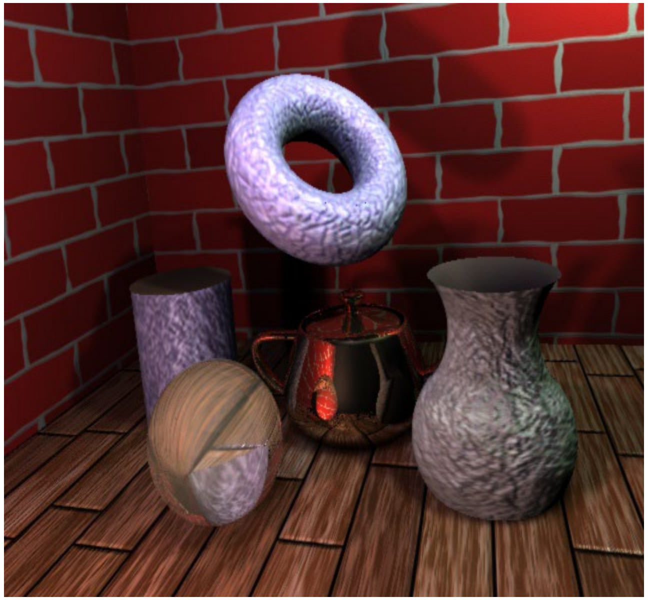
Representations

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

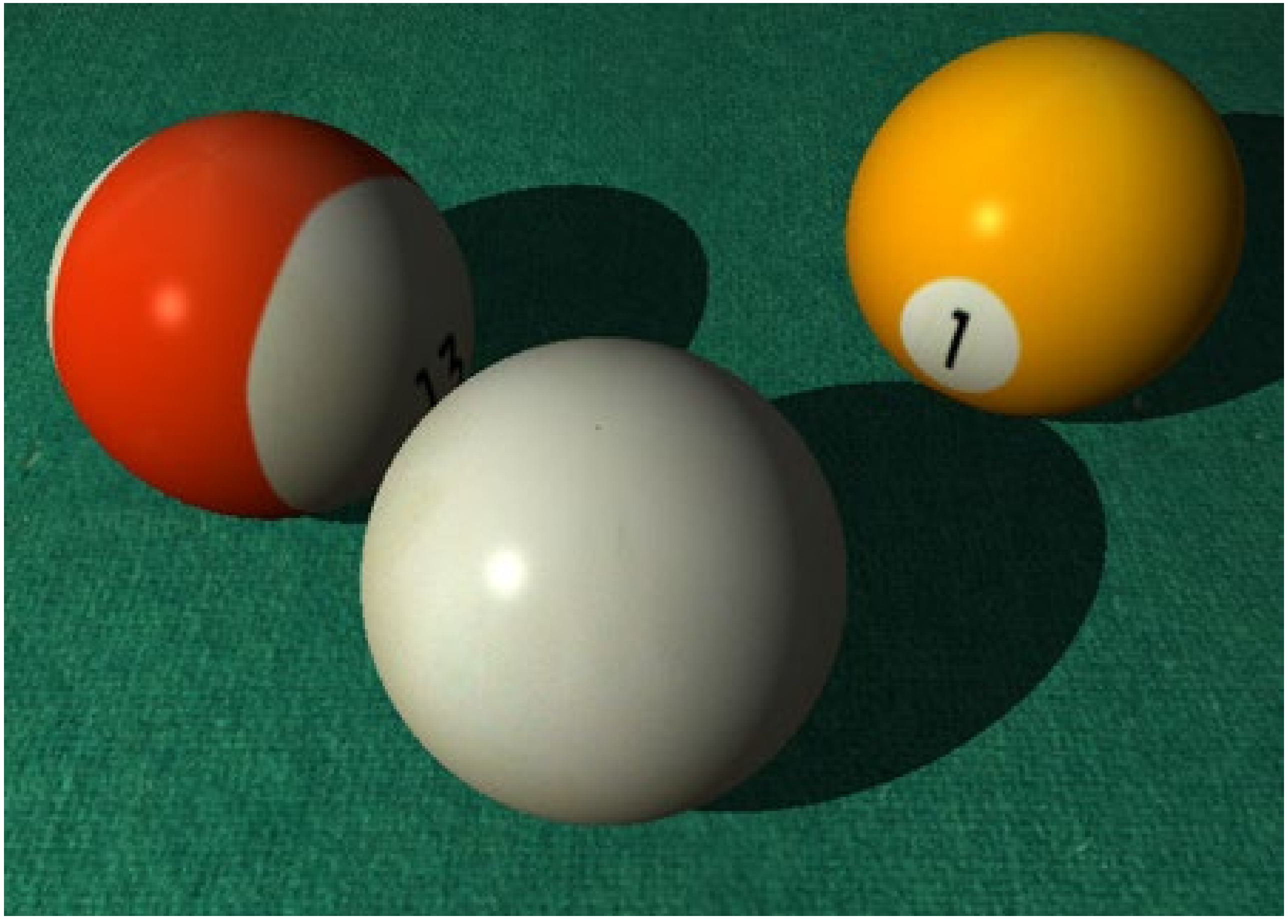
Representations

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

Texture Mapping

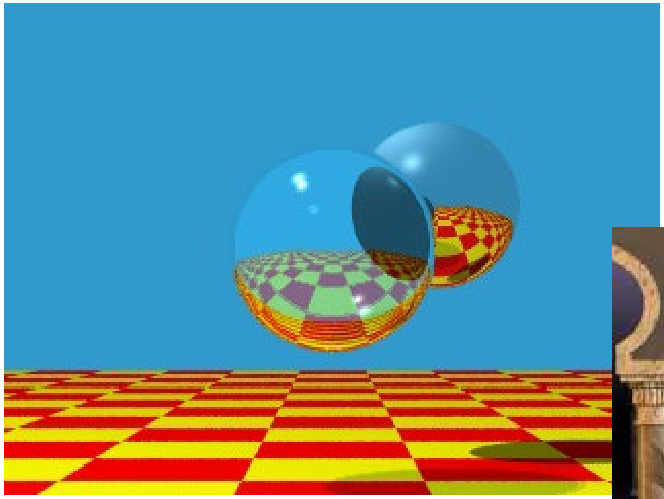


Lighting and Shading

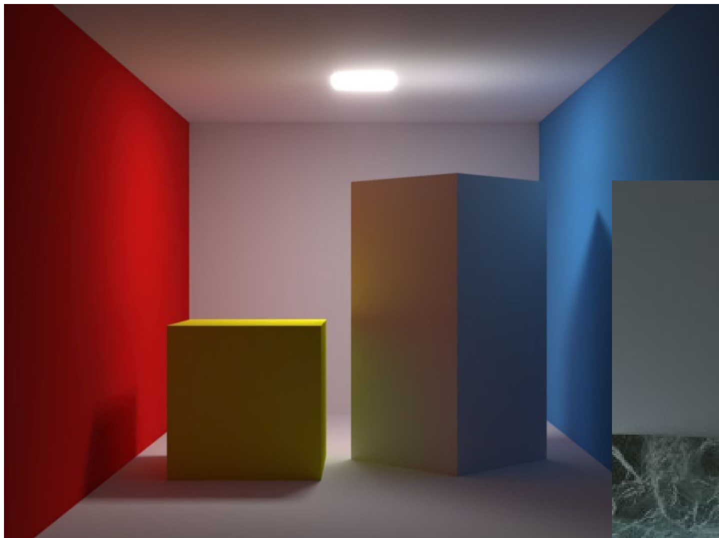


…shadows?

Advanced Topics: Ray tracing



Advanced Topics: Global Illumination



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 >4.0

OpenGL

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

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();
}
```

(Free)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

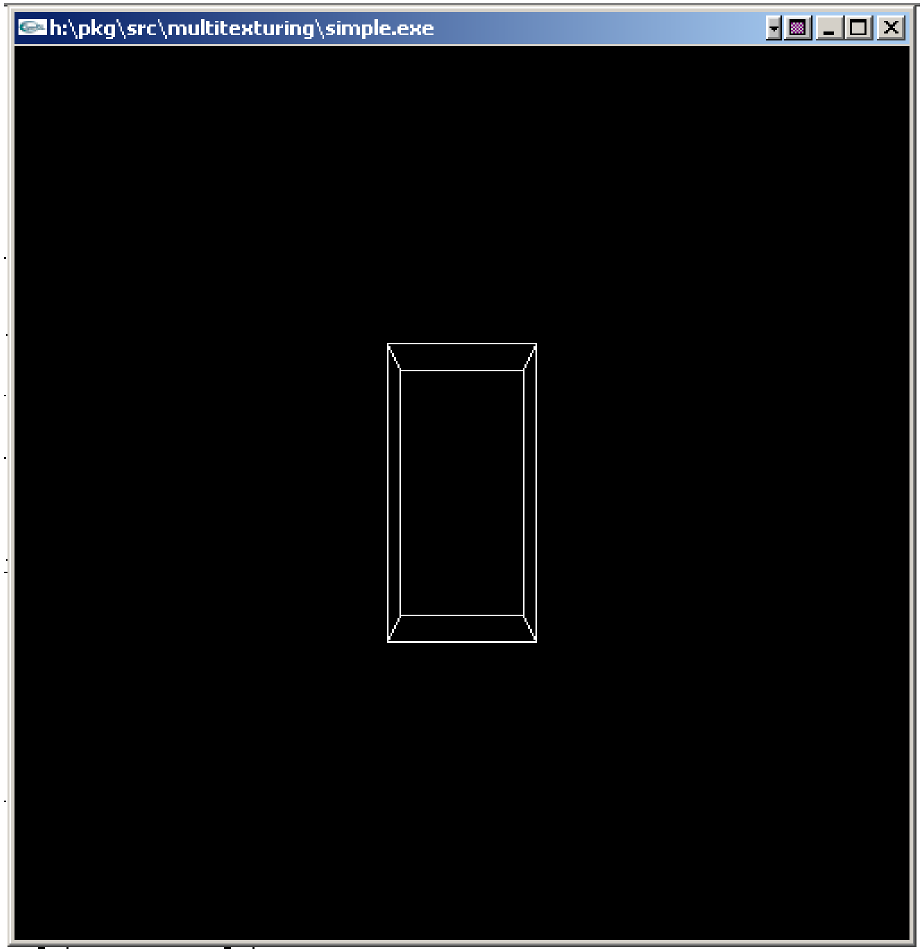
```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

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 ();
}
```

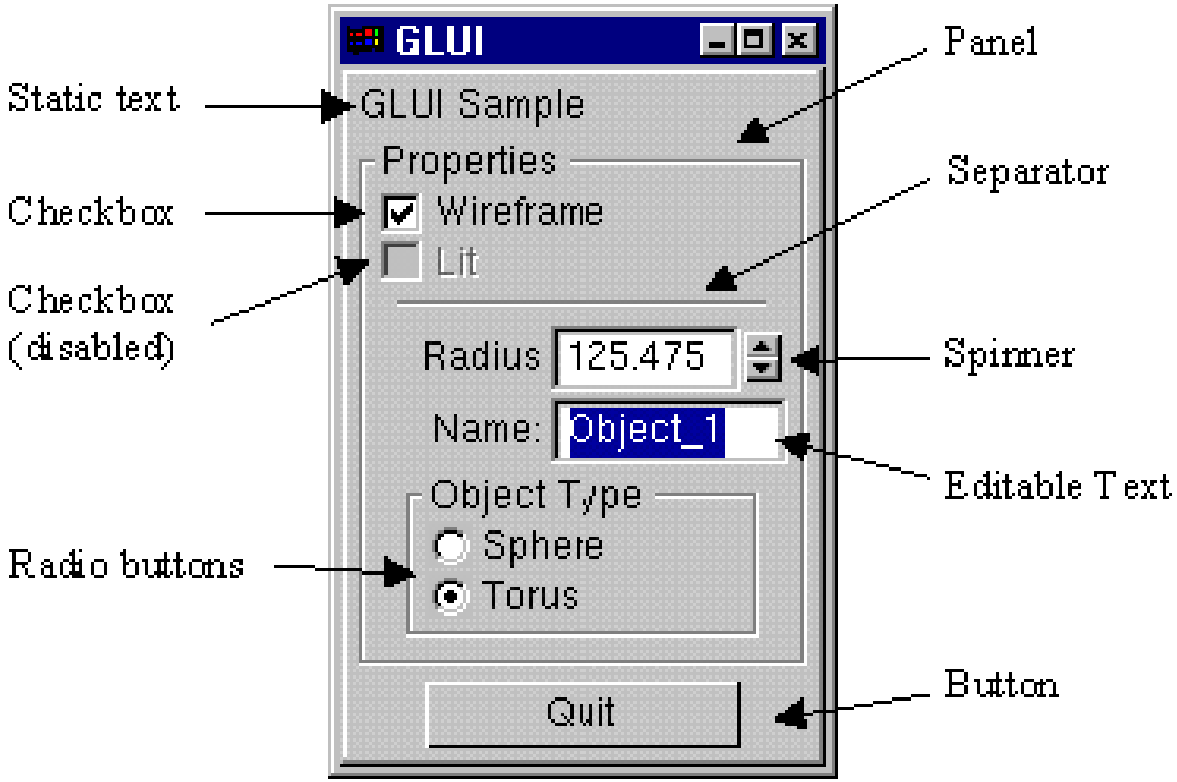
```
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
    glOrtho (-1.5, 1.5, -1.5*(GLfloat)h/(GLfloat)w,
      1.5*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
  else
    glOrtho (-1.5*(GLfloat)w/(GLfloat)h,
      1.5*(GLfloat)w/(GLfloat)h,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



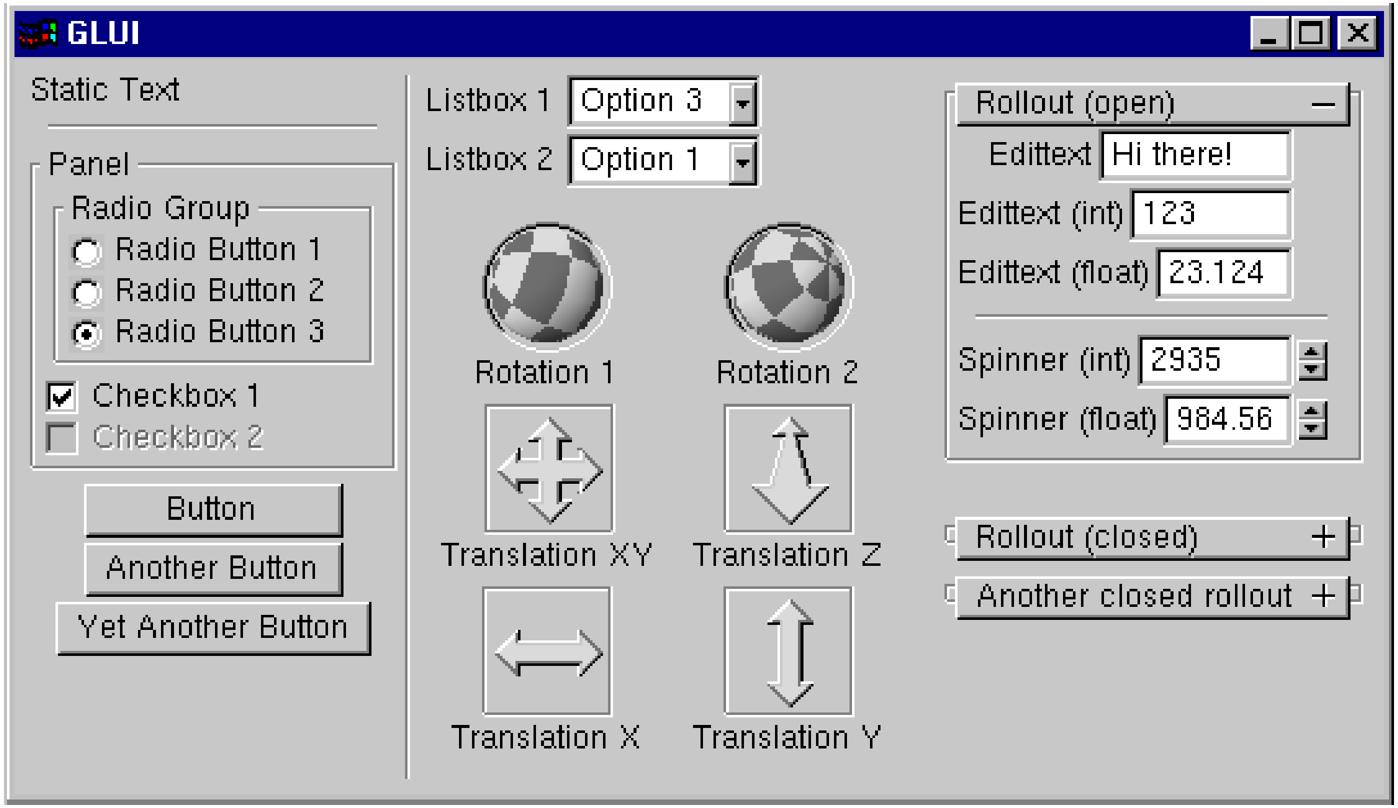
GLUI

- = Graphics Library User Interface



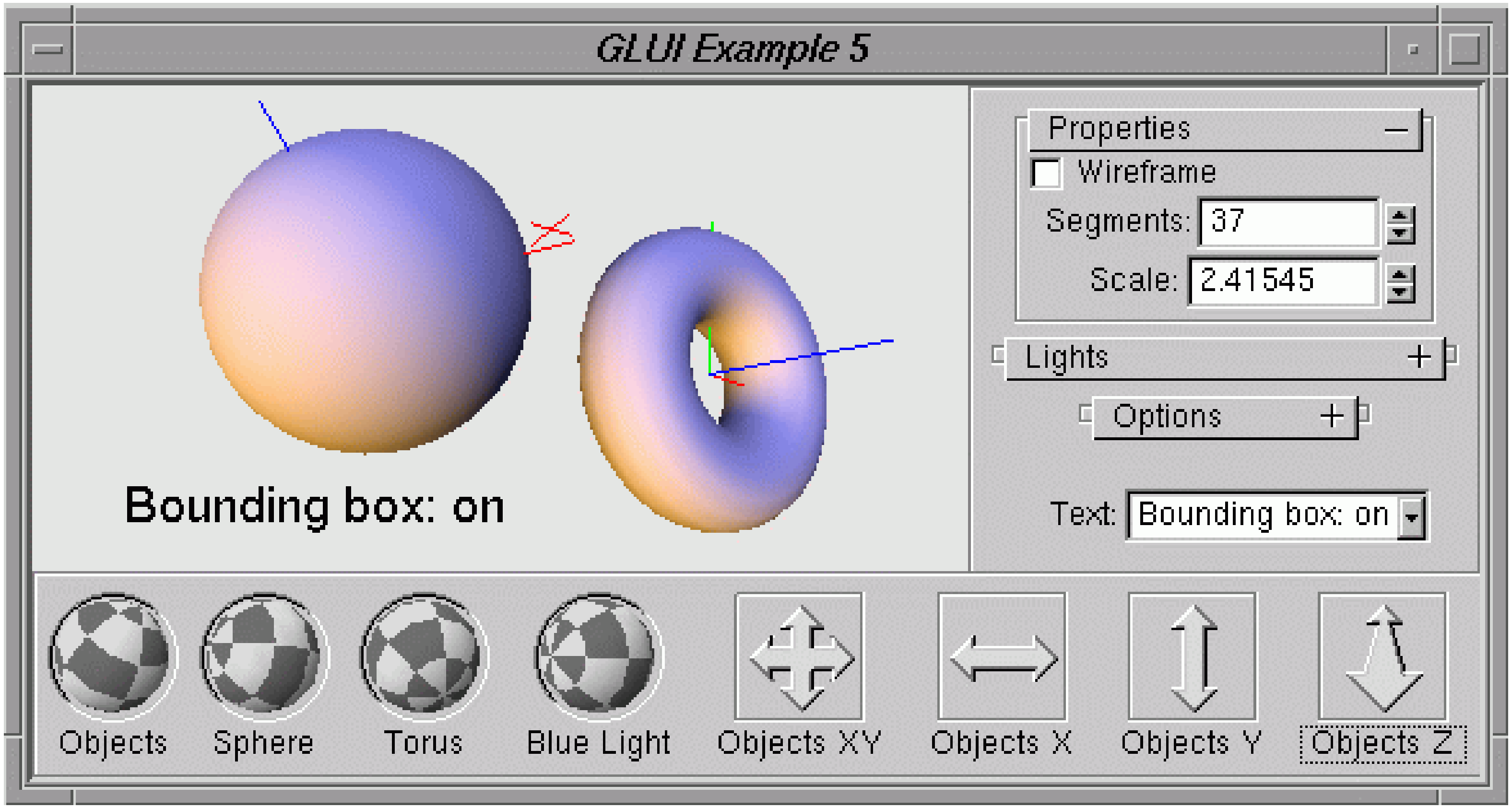
GLUI

- = Graphics Library User Interface



GLUI

- = Graphics Library User Interface



Qt

- Qt is a cross-platform application and UI framework with APIs for C++ programming and Qt Quick for rapid UI creation

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

GLSL

- OpenGL Shading Language
  - Specification
  - Quick reference
  - Example:
    - phong.pix
    - phong.vrt

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