Interactive Computer Graphics
CS 432

Prof. David E. Breen
Department of Computer Science
Objectives

• Introduction to Interactive Computer Graphics
  - Algorithms
  - Mathematics
  - Software
  - Hardware

• Top-down approach

• Shader-based WebGL 2.0
  - Integrates with HTML5
  - Code runs in latest browsers
  - Derived from OpenGL ES 3.0
Credits


• Slides based on lectures for CS/EECE 412 Computer Graphics at the University of New Mexico by Prof. Edward Angel
Prerequisites

• Good programming skills
  - High-level and low-level
• Basic Data Structures
  - Lists
  - Arrays
• Geometry
• Linear Algebra
  - Vectors & matrices
• CS 430 would be helpful, but not required
Requirements

• Weekly Programming Projects

• Go to class web site
Web Resources

• www.cs.unm.edu/~angel
• www.interactivecomputergraphics.com/Code
• www.opengl.org
• www.khronos.org/webgl
• www.chromeexperiments.com/webgl
• webglfundamentals.org
Computer Graphics

- Computer graphics deals with all aspects of creating images with a computer
  - Hardware
  - Software
  - Applications
Example

• Where did this image come from?

• What hardware/software did we use to produce it?
Preliminary Answer

• **Application**: The object is an artist’s rendition of the sun for an animation to be shown in a domed environment (planetarium)

• **Software**: Maya for modeling and rendering but Maya is built on top of OpenGL

• **Hardware**: PC with graphics card for modeling and rendering
Basic Graphics System

Input devices

Processor (CPU) → Graphics processor → Frame buffer

CPU Memory → GPU Memory

Image formed in frame buffer

Output device

- Computer graphics goes back to the earliest days of computing
  - Strip charts
  - Pen plotters
  - Simple displays using A/D converters to go from computer to calligraphic CRT

- Cost of refresh for CRT too high
  - Computers slow, expensive, unreliable
Cathode Ray Tube (CRT)

Can be used either as a line-drawing device (calligraphic) or to display contents of frame buffer (raster mode)
Generic Flat Panel Display

Vertical grid

Light emitting elements

Horizontal grid

- **Wireframe graphics**
  - Draw only lines
- **Sketchpad**
- **Oscilloscope**
- **Display Processors**
- **Storage tube**

wireframe representation of sun object
Sketchpad

• Ivan Sutherland’s PhD thesis at MIT (1963)
  - Recognized the potential of man-machine interaction
  - Loop
    • Display something
    • User moves light pen
    • Computer generates new display
  - Sutherland also created many of the now common algorithms for computer graphics
Display Processor

- Rather than have the host computer try to refresh display use a special purpose computer called a *display processor* (DPU)

- Graphics stored in display list (display file) on display processor

- Host *compiles* display list and sends to DPU

• Raster Graphics
• Beginning of graphics standards
  - IFIPS
    • GKS: European effort
      – Becomes ISO 2D standard
    • Core: North American effort
      – 3D but fails to become ISO standard

• Workstations and PCs
Raster Graphics

- Image produced as an array (the raster) of picture elements (pixels) in the frame buffer
Raster Graphics

- Allows us to go from lines and wire frame images to filled polygons
Although we no longer make the distinction between workstations and PCs, historically they evolved from different roots:

- Early workstations characterized by:
  - Networked connection: client-server model
  - High-level of interactivity

- Early PCs included frame buffer as part of user memory:
  - Easy to change contents and create images

Realism comes to computer graphics

smooth shading  environment mapping  bump mapping

Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015

• Special purpose hardware
  - Silicon Graphics geometry engine
    • VLSI implementation of graphics pipeline

• Industry-based standards
  - PHIGS
  - RenderMan

• Networked graphics: X Window System

• Human-Computer Interface (HCI)

• OpenGL API
• Completely computer-generated feature-length movies (Toy Story) are successful
• New hardware capabilities
  - Texture mapping
  - Blending
  - Accumulation, stencil buffers
Photorealism

Graphics cards for PCs dominate market
- Nvidia, ATI

Game boxes and game players determine direction of market

Computer graphics commonplace in movie industry: Maya, Lightwave

Programmable pipelines

New display technologies
• Graphics is now ubiquitous
  - Cell phones
  - Embedded
• OpenGL ES and WebGL
• Alternate and Enhanced Reality
• 3D Movies and TV
• General Purpose computing on GPUs (GPGPU)
Image Formation
Objectives

• Fundamental imaging notions
• Physical basis for image formation
  - Light
  - Color
  - Perception
• Synthetic camera model
• Other models
• In computer graphics, we form images which are generally two dimensional using a process analogous to how images are formed by physical imaging systems
  - Cameras
  - Microscopes
  - Telescopes
  - Human visual system
Elements of Image Formation

- Objects
- Viewer
- Light source(s)

- Attributes that govern how light interacts with the materials in the scene
- Note the independence of the objects, the viewer, and the light source(s)
• *Light* is the part of the electromagnetic spectrum that causes a reaction in our visual systems

• Generally these are wavelengths in the range of about 350-750 nm (nanometers)

• Long wavelengths appear as reds and short wavelengths as blues
# Photons

<table>
<thead>
<tr>
<th>Wavelength (nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>600</td>
</tr>
<tr>
<td>700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Photons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Violet</td>
</tr>
<tr>
<td>Blue</td>
</tr>
<tr>
<td>Green</td>
</tr>
<tr>
<td>Yellow</td>
</tr>
<tr>
<td>Orange</td>
</tr>
<tr>
<td>Red</td>
</tr>
</tbody>
</table>

---

**Spectral Energy Distributions**

### Laser

- Wavelength (nm)

### Pure day light

- Wavelength (nm)

### Less White (Gray)

- Wavelength (nm)

### White

- Wavelength (nm)
Luminance and Color Images

• Luminance Image
  - Monochromatic
  - Values are gray levels
  - Analogous to working with black and white film or television

• Color Image
  - Has perceptual attributes of hue, saturation, and lightness
  - Do we have to match every frequency in visible spectrum? No!
Three-Color Theory

• Human visual system has two types of sensors
  - Rods: monochromatic, night vision
  - Cones
    • Color sensitive
    • Three types of cones
    • Only three values (the tristimulus values) are sent to the brain

• Need only match these three values
  - Need only three primary colors
Additive and Subtractive Color

- Additive color
  - Form a color by adding amounts of three primaries
    - CRTs, projection systems, positive film
  - Primaries are Red (R), Green (G), Blue (B)
RGB Color Cube

- RGB used in Monitors and other light emitting devices
- TV uses YIQ encoding which is somewhat similar to RGB

http://prometheus.cecs.csulb.edu/~jewett/colors/
Additive and Subtractive Color

- **Subtractive color**
  - Form a color by filtering white light with cyan (C), Magenta (M), and Yellow (Y) filters
    - Light-material interactions
    - Printing
    - Negative film
Color Models CMY

- Describes hardcopy color output
- We see colors of reflected light
- Cyan ink absorbs red light and reflects green and blue
- To make blue, use cyan ink (to absorb red), and magenta ink (to absorb green)

http://prometheus.cecs.csulb.edu/~jewett/colors/
HSB: hue, saturation, and brightness

• Also called HSV (hue saturation value)

• Hue is the actual color. Measured in degrees around the cone (red = 0 or 360, yellow = 60, green = 120, etc.).

• Saturation is the purity of the color, measured in percent from the center of the cone (0) to the surface (100). At 0% saturation, hue is meaningless.

• Brightness is measured in percent from black (0) to white (100). At 0% brightness, both hue and saturation are meaningless.
One way to form an image is to follow rays of light from a point source finding which rays enter the lens of the camera. However, each ray of light may have multiple interactions with objects before being absorbed or going to infinity.
Global vs Local Lighting

- Cannot compute color or shade of each object independently
  - Some objects are blocked from light
  - Light can reflect from object to object
  - Some objects might be translucent
Synthetic Camera Model

- Local lighting
- Projects geometry onto image plane.
- Use local info to shade point

Local lighting projects geometry onto the image plane. Use local information to shade points.
Pinhole Camera

Use trigonometry to find projection of point at \((x,y,z)\)

\[
\begin{align*}
  x_p &= -x/(z/d) \\
  y_p &= -y/(z/d) \\
  z_p &= d
\end{align*}
\]

These are equations of simple perspective
Advantages – Local Lighting

• Separation of objects, viewer, light sources
• Two-dimensional graphics is a special case of three-dimensional graphics
• Leads to simple software API
  - Specify objects, lights, camera, attributes
  - Let implementation determine image
• Leads to fast hardware implementation
• Allows for pipelined & parallel computations
Why not ray tracing?

Ray tracing seems more physically based, so why don’t we use it to design a graphics system?

• It is actually simple for simple objects, such as polygons and quadrics with simple point sources.

• In principle, can produce global lighting effects such as shadows and multiple reflections, but ray tracing is slow and not well-suited for interactive applications.

• Ray tracing hardware has been developed (Pixel Machines), but has not survived in market.

• Real-time ray tracing is recently possible with GPUs (NVIDIA Quadro RTX)!
Models and Pipeline Architecture
Objectives

• Learn the basic design of a graphics system
• Introduce pipeline architecture
Image Formation Revisited

- Can we mimic the synthetic camera model to design graphics hardware & software?

- Application Programmer Interface (API)
  - Need only specify
    - Objects
    - Materials
    - Viewer
    - Lights

- But how is the API implemented?
Physical Approaches

• **Ray tracing**: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity
  - Can handle some global effects
    • Multiple reflections
    • Translucent objects
  - Slow
  - Must have whole data base available at all times

• **Radiosity**: Energy-transfer-based approach
  - Very slow for dynamic scenes
Practical Approach

- Process objects one at a time in the order they are generated by the application
  - Can consider only local lighting
- Pipeline architecture
  - All steps can be implemented in hardware on the graphics card

application program display

WC – World Coords  CC – Camera Coords  NPC – Normalized Projection Coords  SC – Screen Coords
Vertex Processing

• Much of the work in the pipeline is in converting object representations from one coordinate system to another
  - Object coordinates
  - Camera (eye) coordinates
  - Screen coordinates
• Every change of coordinates is equivalent to a matrix transformation
• Vertex processor also computes vertex colors
Projection

• *Projection* is the process that combines the 3D view with the 3D objects to produce the 2D image
  - Perspective projections: all projectors meet at the center of projection
  - Parallel projection: projectors are parallel, center of projection is replaced by a direction of projection
Planar Geometric Projections

- **Projections onto Planes**
  - Consider the line $AB$

- **Perspective Projection**
  - a single viewing location
  - similar to a photograph

- **Parallel Projection**
  - viewing location at $\infty$
  - good for capturing shape and dimensions
Primitive Assembly

Vertices must be collected into geometric objects before clipping and rasterization can take place:
- Line segments
- Polygons
- Curves and surfaces
Clipping

Just as a real camera cannot “see” the whole world, the virtual camera can only see part of the world or object space.

- Objects that are not within this volume are said to be clipped out of the scene.
Rasterization

- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors.
- Rasterizer produces a set of fragments for each object.
- Fragments are "potential pixels" - Have a location in frame buffer - Color and depth attributes
- Vertex attributes are interpolated over objects by the rasterizer.

Diagram:
- Vertices → Vertex Processor → Clipper and Primitive Assembler → Rasterizer → Fragment Processor → Pixels
Fragment Processing

• Fragments are processed to determine the color of the corresponding pixel in the frame buffer
• Colors can be determined by texture mapping or interpolation of vertex colors
• Fragments may be blocked by other fragments closer to the camera
  - Hidden-surface removal
Pipeline Summary

- **Vertex Processor (executes vshader.glsl)**
  - Input: 3D vertices in world coordinates
  - Output: 3D vertices in clip (camera) coordinates
  - Transforms and projects vertices
    Could also compute colors

- **Primitive Assembler and Clipper**
  - Output: 2D Clipped geometry
    (points, lines & triangles)
  - Assembles vertices into geometry and clips it against canonical view volume
Pipeline Summary

• Rasterizer
  - Input: 2D Geometry that was inside view volume
  - Output: Fragments (pixels with other information)
  - Scan converts 2D geometry into fragments in window coordinates
  - Interpolates other info across the geometry

• Fragment Processor (executes fshader.glsl)
  - Output: Pixel colors
  - Uses fragment info to compute color at pixel (i,j)
The Programmer’s Interface

- Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)
API Contents

• Functions that specify what we need to form an image
  - Objects
  - Viewer
  - Light Source(s)
  - Materials

• Other information
  - Input from devices such as mouse and keyboard
  - Capabilities of system
Object Specification

• Most APIs support a limited set of primitives including
  - Points (0D object)
  - Line segments (1D objects)
  - Polygons (2D objects)
  - Some curves and surfaces
    • Quadrics
    • Parametric polynomials

• All are defined through locations in space or vertices
Example (old style)

\begin{align*}
&\text{type of object} \\
&\text{location of vertex} \\
&\text{end of object definition}
\end{align*}

\begin{verbatim}
glBegin(GL_POLYGON) \\
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glVertex3f(0.0, 0.0, 1.0);
glEnd();
\end{verbatim}
• Put geometric data in an array
  
  ```javascript
  var points = [
    vec3(0.0, 0.0, 0.0),
    vec3(0.0, 1.0, 0.0),
    vec3(0.0, 0.0, 1.0),
    vec3(0.0, 0.0, 1.0),
  ];
  ```

• Send array to GPU

• Tell GPU to render as triangle
Camera Specification

• Six degrees of freedom
  - Position of center of lens
  - Orientation
• Lens
• Film size
• Orientation of film plane
• Types of lights
  - Point sources vs. distributed sources
  - Spot lights
  - Near and far sources
  - Color properties
Materials

• Material properties
  - Absorption: color properties
  - Scattering
    • Diffuse, Specular