An Interactive Introduction to OpenGL Programming

What You’ll See Today
- General OpenGL Introduction
- Rendering Primitives
- Rendering Modes
- Lighting
- Texture Mapping
- Additional Rendering Attributes
- Imaging

Goals for Today
- Demonstrate enough OpenGL to write an interactive graphics program with
  - lighting
  - texture mapping
  - introduce advanced topics for future investigation

Related APIs
- AGL, GLX, WGL
  - part of OpenGL
  - Windows or Mac O/S
  - portable windowing API
- GLUT (OpenGL Utility Toolkit)
  - part of OpenGL
  - glue between OpenGL and windowing systems
  - not officially part of OpenGL

OpenGL and Related APIs

Sample Program
```c
#define GL_ALL_ATTRIB_BITS
#define GL_ALL_ATTRIB_BITS

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

int main(int argc, char** argv)
```

OpenGL Initialization
- Setup whatever state you’re going to use
  ```c
  void main(void)
  ```
  ```c
  void main(void)
  ```

GLUT Callback Functions
- Routine to call when something happens
  ```c
  void display(void)
  ```
  ```c
  void display(void)
  ```
Controlling Rendering Appearance
- From Wireframe to Texture Mapped

OpenGL’s State Machine
- All rendering attributes are encapsulated in the OpenGL State
- Rendering pipeline
- Shading
- Lighting
- Texture mapping

Void idle( )

t { 
    glutIdleFunc( display ); 
    glutIdleFunc( idle ); 
}

Manipulating OpenGL State
- Appearance is controlled by current state
  - set each primitive to render
  - light state
  - Manipulating vertex attributes in sort context may be manipulated state
    - glVertex3f( x, y, z )
    - glLightModelf( p, v )

OpenGL Buffers
- Elementary Rendering
- Geometry Primitives
- Color or index

OpenGL Color Models
- RGBA or color index

Transforms in OpenGL
- Modeling
- Viewing
- Projection
- Animation
- Map to screen
An Interactive Introduction to OpenGL Programming

**Camera Analogy**
- 3D is just like taking a photograph (lots of photographs!)

**Transformations**
- Specification of transformations
- Combine (composite) transformations
- Camera Analogy and Transformations
  - Projection transformations: adjust the lens of the camera
  - Viewing transformations: specify world position and orientation of the viewing window
  - Modeling transformations: specify world coordinates
  - Viewport transformations: arrange or resize the physical photograph

**3D Transformations**
- A vertex is transformed by a 4 x 4 matrix
  - All affine operations are linear multiplications
  - Matrices are always post-multiplied
  - Product of matrix and vector is a transformation or resulting image

**Matrix Operations**
- Specify Corrected Matrix Block
  - Other Basic Block Operations
  - Viewport

**Projection Tutorial**
- Shape of viewing frustum
  - Perspective projection
  - Orthographic projection
  - Tangent projection

**Transformation Tutorial**
- Moving object
  - Rotate object around arbitrary axis
  - Dilate (stretch or shrink) or mirror object

**Homogeneous Coordinates**
- Each vertex is a 4 vector

**Coordinate Systems and Transformations**
- Steps in Forming an Image
  - Specify geometry (world coordinates)
  - Project (window coordinates)
  - Map to viewport (screen coordinates)
  - Each step uses transformations

**Affine Transformations**
- Want transformations which preserve geometry
  - Lines, parallelograms, rectangles
  - Affine = line preserving
  - Rotation, translation, scaling
  - Position
  - Concatenate (composition)

**Specifying Transformations**
- Programmer has two styles of specifying transformations
  - Specify a matrix (glLoadMatrix, glMultMatrix)
  - Specify operations (glLoadIdentity, glPushMatrix)
  - Programmer does not have to remember the exact matrices
  - Make appropriate use of new block (programming style)

**3D Transformations**
- Prior to rendering, view, locate, and orient:
  - Eyecenre, position
  - 3D geometry
  - Manage the matrices
  - Including matrix stack
  - Combine (composite) transformations
Connection: Viewing and Modeling
- Viewing camera is equivalent to moving every object in the world towards a stationary camera.
- Viewing transformations are equivalent to several modeling transformations.
- Viewing is not consistent.
- Everything else is right-handed, including the view to be rendered.

Projection transversals (gluPerspective, gluLookTo) are left handed.
- All objects and other things from view plane.

Common Transformation Usage
- 3 examples of resize():
  - register: projection & viewing transformation.
  - Usually called when window is resized.
  - Usually called when window is resized.

Projection & Translate
- Some effect on previous LookAt:
  - gliLoadIdentity();
  - gluMatrixMode( GL_PROJECTION );
  - gluViewport( 0, 0, (GLsizei) w, (GLsizei) h );
  - gluLookAt( 0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0 );

Reversing Coordinate Transformations
- Problem 1: Hierarchical objects
  - one position depends upon a previous position.
  - Position at root level.
  - Solution 1: Moving local coordinate system.
- Position at root level.
- Problem 2: Objects move relative to absolute world origin.
- All objects are relative.
- Solution 2: Fixed coordinate system.
- Position at root level.
- Position at root level.
- Position at root level.
- Position at root level.

Additional Clipping Planes
- At least 6 more clipping planes available.
- Good for cross-sections.
- ModelView matrix moves clipping planes.
- gluClipPlane( GL_FRONT, GL_CLIP_PLANE0, ... );
- gluClipPlane( GL_FRONT, GL_CLIP_PLANE1, ... );
- gluClipPlane( GL_FRONT, GL_CLIP_PLANE2, ... );
- gluClipPlane( GL_FRONT, GL_CLIP_PLANE3, ... );
- gluClipPlane( GL_FRONT, GL_CLIP_PLANE4, ... );
- gluClipPlane( GL_FRONT, GL_CLIP_PLANE5, ... );

Animation and Depth Buffering
- Discuss double buffering and animation.
- Discuss hidden surface removal using the depth buffer.
- Discuss moving objects.
- GluClipPlane.

Depth Buffering and Hidden Surface Removal
- Request a double buffered color buffer.
- Clear color buffer.
- Render scene.
- Request swap of front and back buffers.
- Render scene.
- Repeat steps 2-4 for animation.
Lighting

Lighting simulates how objects reflect light:
- Global light parameters
- Global lighting parameters
- Position and type
- Use and normals for proper lighting

Surface Normals
Normals define how a surface reflects light:
- OpenGL supports two types of lights:
- Use attenuation to control light intensity
- Surface smoothness
- Light color properties
- Global light properties
- Local position and type

Turning on the Lights
- Turn on the power
- Push and pop matrices to uniquely control a light's position
- Global light properties
- Local light properties
Rendering Fonts using Bitmaps

OpenGL uses bitmaps for font rendering
- Using bitmaps to display text
- Bitmaps can be stored in memory
- Fonts can be loaded into OpenGL for rendering

Tips for Better Lighting

- Use a single light source for maximum lighting
- Separate specular and diffuse lighting
- Use global ambient color
- Use local viewer mode
- Use spotlights

Advanced Lighting Features

- Light attenuation
- Sprawl lighting effects
- Global ambient color
- Local viewer mode
- Separate specular color

Light Model Properties

- GL_LIGHT_MODEL_AMBIENT
- GL_LIGHT_MODEL_LOCAL_VIEWER
- GL_LIGHT_MODEL_COLOR_CONTROL

Pixel-based primitives

- Bitmaps
- Images
- Framebuffer pixel copy
- Programmable pixel storage

Programmable pixel storage and transfer operations

- glPixelZoom
- glCopyPixels
- glCopyTexImage

Demonstrate how to get OpenGL to read bitmaps and image rectangles

Describe OpenGL's raster primitives:
- reading pixels
- reading from framebuffer
- rendering pixel rectangles

Tips for Better Lighting

- Recall lighting computed only at vertices
- Use a single infinite light for faster lighting
- Avoid LOD lighting

Positioning Image Primitives

- glRasterPos
- glBitmap

Removing Images

- glBitmap
- glCopyImage

Texture Mapping

- Texture mapping
- Texture coordinates

Storage and Transfer Modes

- Storage modes
- Transfer modes

Rendering Images

- glBitmap
- glCopyImage
- glCopyTexImage

Reading Images

- glReadPixels
- glCopyPixels

Reading Pixels

- glReadPixels
- glCopyPixels

Pixel Pipeline

- Programable pixel storage
- Rasterization and transfer operations
- Texture map

Programmable pixel storage

- glPixelZoom
- glCopyPixels
- glCopyTexImage

Demonstrate how to get OpenGL to read bitmaps and image rectangles

Describe OpenGL's raster primitives:
- reading pixels
- reading from framebuffer
- rendering pixel rectangles

Tips for Better Lighting

- Recall lighting computed only at vertices
- Use a single infinite light for faster lighting
- Avoid LOD lighting
An Interactive Introduction to OpenGL Programming

**Texture Mapping and the OpenGL Pipeline**
- Images and geometry flow through separate pipelines that join at the rasterizer.
- "Complex" textures do not affect geometric complexity.

**Texture Example**
The texture (below) is a 256 x 256 image that has been mapped to a rectangular polygon which is viewed in perspective.

**Texture Application Methods**
- Generating Texture Coordinates
  - Automatically generate texture coordinates based upon distance from plane.
  - Specify plane
  - Set generation modes
  - Textures can also be specified to avoid division by distance.

**Converting A Texture Image**
- If dimensions of image are not power of 2, dimensions of image are powers of 2.
- Generate texture coordinates based upon distance from plane.
- Texture coordinates can also be specified to avoid division by distance.

**Specifying a Texture: Other Methods**
- Use frame buffer as source of texture image
- Modify part of a defined texture
- Do both with glGetUniformLocation( )

**Texture Objects**
- Create texture objects with texture data and state.
- Generate texture before using glTexImage2D() or glTextureImage() with texture object.

**Texture Objects (cont.)**
- Texture objects can be shared by several graphics contexts.
- Textures can be specified to be specular or diffuse.

**Filter Modes**
- Filter modes: blend, modulate, or replace texels.
- Wrapping, filtering.

**Filtering**
- Special mipmaps for filtering.
- Special mipmap minification filters.
- Special mipmap magnification.

**Texture Functions**
- Textures can be specified to be specular or diffuse.
- Blended, modulated, or replaced.
Display Lists

- Creating a display list
  - Push the list name onto the display list stack
  - Render a display list
  - Pop the display list stack

Display Lists

- Not all OpenGL routines can be stored in display lists
- State changes persist, even after a display list is finished
- Display lists can call other display lists
- Display lists are not executable, but you can take
- Call a created list

Immediate Mode versus Display Lists

- Immediate Mode Graphics
  - Render in-memory display lists and display right away
  - In-place modification of objects
- Display List Graphics
  - Push the display list onto the display list stack
  - Display list
  - Can be shared among OpenGL graphics contexts

Immediate Mode versus Display Listed Rendering

- Immediate Mode
  - Rendering and using immediate mode display lists
- Display Lists
  - Pushing and rendering display lists

Display Lists and Hierarchy

- Consider model of a car
  - Create display list for spheres
  - Create display list for wheels
  - Add other display lists to car

Display Lists

- Not all OpenGL routines can be stored in display lists
- State changes persist, even after a display list is finished
- Display lists can call other display lists
- Display lists are not executable, but you can take
- Call a created list

Display Lists

- Not all OpenGL routines can be stored in display lists
- State changes persist, even after a display list is finished
- Display lists can call other display lists
- Display lists are not executable, but you can take
- Call a created list
**Frag Plane**

- **Frag Plane in Focus**
  - Jitter the viewer to keep one plane unchanged
  - Functions:
    - Focus
    - Depth of Field
    - Motion Blur

- **Depth Cueing**
  - Combine pixels with what's in already in the frame buffer
  - glBlendFunci( src, dst )

  - Blending:
    - Problem of compositing into color buffers
    - Source:
      - GL_ADD, GL_MULTI
    - Destination:
      - GL_SRC_ALPHA

- **Antialiasing**
  - Alpha: the 4th Color Component
  - Antialiasing
  - Jitter the viewer to keep one plane unchanged

- **Environmenta l effects**
  - Blended pixel with what's in already in the frame buffer
  - glAccum( op, value )

- **Blending Equation**
  - glBlendFunci( src, dst )
  - Blending
    - Problem:
      - Accumulation buffer acts as a "floating point" color buffer
      - Combine pixels with what's in already in the frame buffer

- **Selection Name Routines**
  - Selection mode:
    - Method to determine which primitives are inside the viewing volume
    - Need to set up a buffer to house results returned to you

- **Selection Mode**
  - Each time we move the viewer, the image shifts
  - Allows for hierarchies of primitives

- **Feedback Mode**
  - Picking is a special case of selection
  - Pick Mode:
    - Multi-pass rendering
    - Antialiasing
    - Blending
  - Selection mode:
    - Method to determine which primitives are inside the viewing volume
    - Need to set up a buffer to house results returned to you
Dithering
- Dither colors for better looking results
  - Used to simulate more available colors

Logical Operations on Pixels
- Combine pixels using bitwise logical operations
  - Constant booleans
    - $\text{and}$
    - $\text{or}$
    - $\text{not}$
    - $\text{xor}$

Using Stencil Mask
- glStencilFunc (func, ref, mask)
  - func: is one of standard comparison functions
  - ref: is the reference value
  - mask: affects in the stencil buffer

Controlling Stencil Buffer
- glStencilPosition (x, y, w, h)
  - x, y: specify the corner of the stencil area
  - w, h: size of the stencil area

Creating a Mask
- glDrawPixels (x, y, w, h, GL_STENCIL, buf)
  - Draws a mask in the stencil buffer
  - Use glDrawPixels to draw objects where stencil = 1

Additional Clipping Test
- glScissor (x, y, w, h)
  - Clips to a specified region of the viewport
  - Useful for updating a small section of a viewport

Picking Ideas
- For OpenGL Picking Mechanism
  - "Why are pixels what is possible (e.g., don't slope)
  - Surely, "Bucket" filled message, instead of bit
  - Several primitives (draw in picking region) tend to
  - Point value to determine which primitives are drawn

Picking Template (cont.)
- •• draw objects where stencil = 1
  - glStencilFunc (GL_EQUAL, 0x1, 0x1)
  - glDisable (GL_ALPHA_TEST)
  - Use gloadName() to name a fragment

Used for Updating a Small Section of a viewport

Alpha Test
- Rejection of pixels based on their alpha value
  - glAlphaFunc (func, ref)
  - glAlphaFunc (GL_LESS, value)

Advanced Imaging
- Imaging Subset
  - Only available if GL_ARB_imaging defined
  - Use glDrawPixels to draw objects where stencil = 1

Summary / Q & A
- Dave Shreiner
- Vicki Shreiner
Thanks for Coming

Questions and Answers

Dave Shreiner  shreiner@sgi.com
Ed Angel  angel@cs.unm.edu
Vicki Shreiner  vshreiner@sgi.com