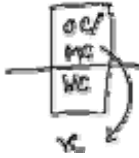


Lecture 19 of 42

Scene Graph Traversal



William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course pages: <http://snipurl.com/1y5gc>

Course web site: <http://www.kddresearch.org/Courses/CIS636>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Readings:

Sections 4.4 – 4.6, Eberly 2^o – see <http://snurl.com/1ye72>



3D Rendering in OpenGL

David Blumenthal, SkyJuggler Consulting

David Murray

Anthony Magro

20 February 2006





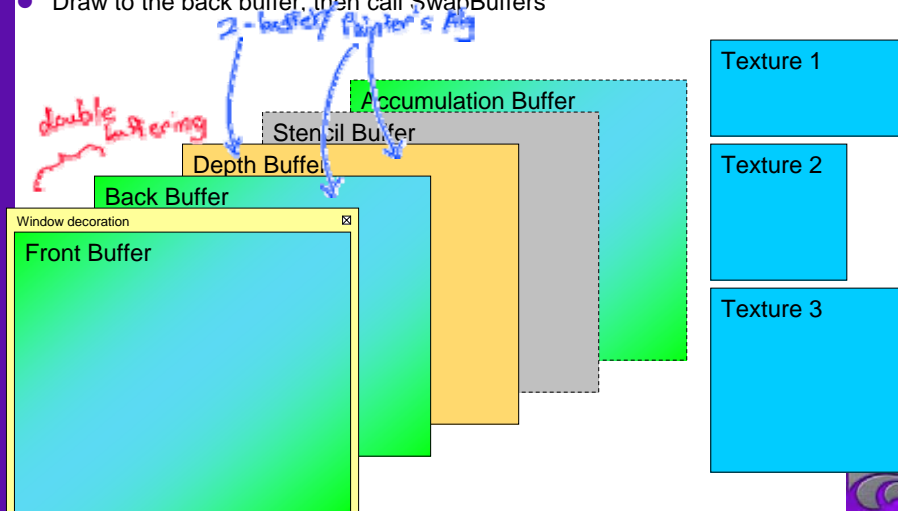
Looking at the Code

- Setup OpenGL State
- Geometry precomputation
- Frame Loop
- Drawing Code



Draw Buffers

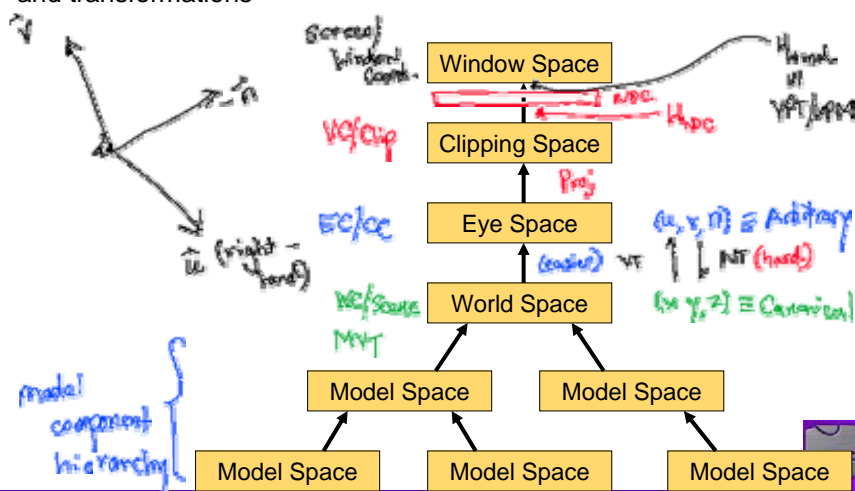
- The Front Buffer is what you see on screen – everything else is off-screen
- Draw to the back buffer, then call SwapBuffers





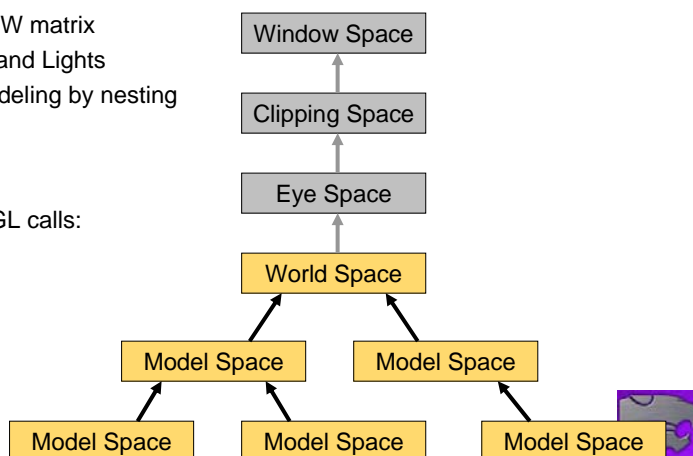
OpenGL Coordinate Spaces

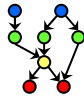
- Model vertices become window coordinates via a series of spaces and transformations



Modeling Transformations

- Use any transformations you want to place model geometry in the "World" space.
- GL_MODELVIEW matrix
- Define Objects and Lights
- Hierarchical Modeling by nesting transforms
 - * glPushMatrix
 - * glPopMatrix
- Most common GL calls:
 - * glTranslate
 - * glRotate





Order is Important

- `glLoadIdentity()`



- `glRotated(45, 0, 0, 1)`



- `glTranslated(5, 0, 0)`



- `glLoadIdentity()`



- `glTranslated(5, 0, 0)`



- `glRotated(45, 0, 0, 1)`



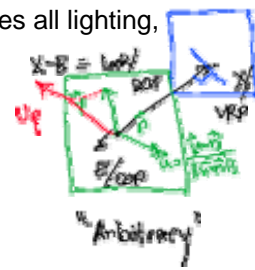
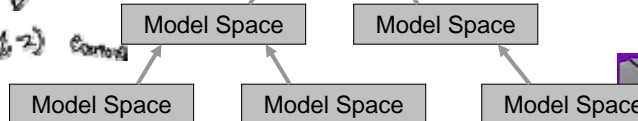
OpenGL commands successively define new "local" coordinate spaces in terms of the "current" or previous local space.



Viewing Transformation

- OpenGL doesn't care about "World" space, it does all lighting, culling, etc. calculations in Eye Space.
- `GL_MODELVIEW` matrix
- Position the world in front of the camera.
- Define Objects and Lights which are relative to the viewpoint
 - * Headlights
 - * First Person objects

(x, y, z) World
 \downarrow View
 (x', y', z') Camera





Positioning the View

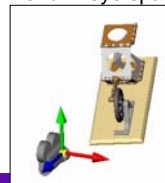
- Don't position the camera in the world...
 - * Position the world in front of the camera!
- Transform the world into "Eye Space"
 - * The camera is at the origin
 - * Looking down the negative Z axis
 - * X points right, Y points up
- gluLookAt
 - * Converts a world space camera into rotation and translation
- Or, invert a camera position matrix

NT



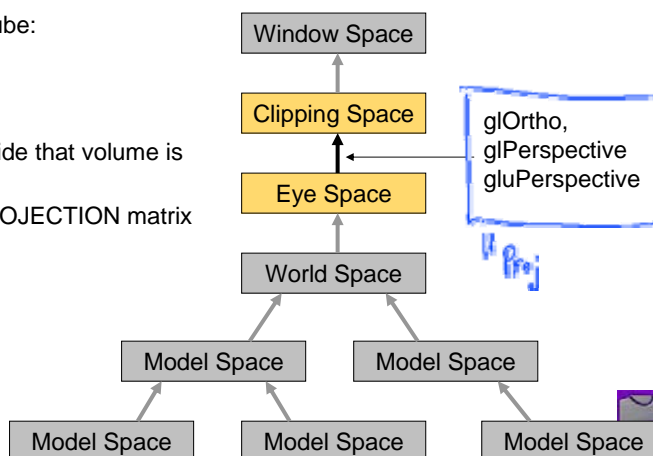
Camera in world space

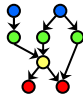
World in eye space



Projection Transformation

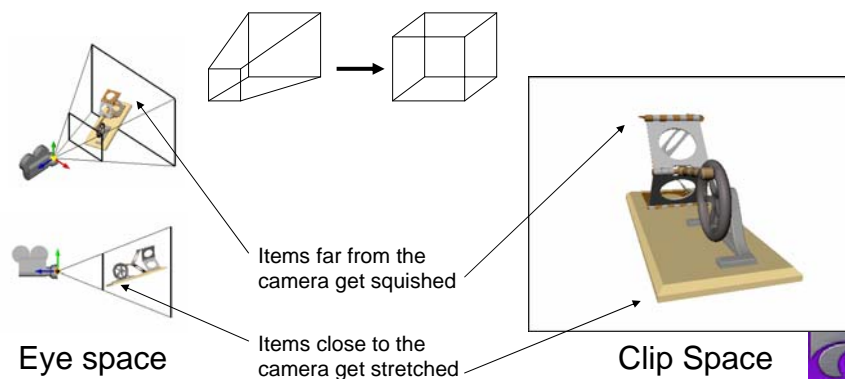
- Transform the region to be displayed into the region of the Clipping Cube.
- The Clipping Cube:
 - * -1 to 1 in X
 - * -1 to 1 in Y
 - * -1 to 1 in Z
- Everything outside that volume is clipped out.
- Use the GL_PROJECTION matrix





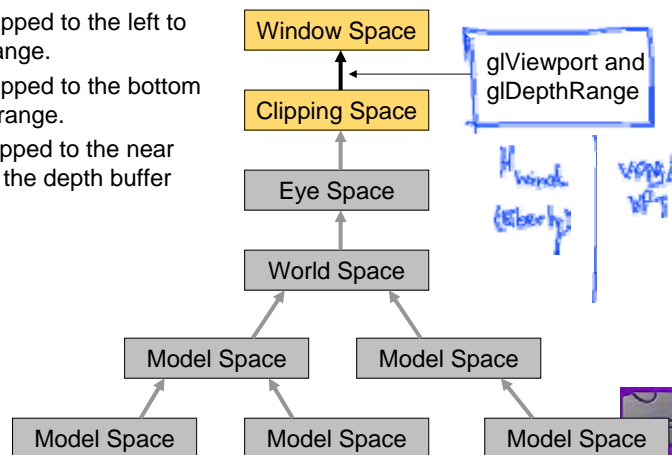
Perspective Projection

- The `GL_PROJECTION` matrix maps a region of eye space to the clipping space cube.
- `glFrustum` creates a non-linear mapping, yielding a perspective effect.



Viewport Transformation

- The Clipping Cube is mapped to the viewport bounds.
- $[-1,1]$ in X is mapped to the left to right viewport range.
- $[-1,1]$ in Y is mapped to the bottom to top viewport range.
- $[-1,1]$ in Z is mapped to the near and far limits of the depth buffer range.





OpenGL Setup

- Create a Window and a GL Context (glut, SDL)
- Load Models, Textures, and Shaders
 - * Load from files, or generate at runtime
 - * Put static geometry into display lists or Vertex Buffer Objects
 - * Put textures onto the card or in Pixel Buffer Objects
- Setup Lighting
 - * Light positions may change, but light colors, ambient lighting, and other lighting parameters often don't
- Setup any other static OpenGL state
 - * Enable depth testing, antialiasing, backface culling
- Initialize the program and animation state



Scene Graphs

- Organize all your models into a structure
 - * Update the structure for animation
 - * Iterate over the nodes to draw
- Create a base SG Node class
 - * Manage tree structure (parent, children pointers)
 - * Manage transforms and bounding boxes
- Subclass for different types of objects
 - * Primitives which can generate their own geometry
 - * Generic Mesh class for loaded geometry
- Lights and Cameras are special nodes
 - * Need to be handled outside the normal traversal





Scene Graphs (continued)

- Keep the actual geometry in a separate class
 - * Contain the logic for managing display lists/VBOs
 - * Allows reuse of geometry for multiple objects
 - * Special logic for drawing at a reduced level of detail
- Keep the appearance in a separate class
 - * Material parameters, texture and shader IDs
 - * Be able to iterate over appearances, and find all the objects which use the same appearance
 - * Special logic for drawing transparent surfaces
- Keep the behavior in a separate class
 - * Behaviors often belong to an object, but are updated by a separate behavior manager



Don't Reinvent the Wheel

- Freely available math libraries:
 - * Wild Magic (<http://www.geometrictools.com/>)
- Freely available image libraries:
 - * libjpeg
 - * libpng
- Model formats and loading libraries:
 - * Plenty of them out there if you search





Performance

- Use the best vertex mechanism you can
 - * Vertex Buffer Objects are the best because the data is in memory which the hardware can access
 - * Vertex arrays are good – several extensions make them better
 - * glVertex3fv is better than glVertex3f
- Don't use glScale if you're using lighting
 - * Requires GL_NORMALIZE to be enabled
- Don't use GL_POLYGON
 - * It's the same thing as GL_TRIANGLE_FAN, but may not be as well optimized



Performance 2

- Display Lists are making a comeback
 - * They used to be for getting data across an XWindows socket.
 - * Now, OpenGL drivers are taking advantage of them to optimize the way the hardware is used.
- Minimize draw calls
 - * Group together objects which use the same textures, shaders, other material properties
 - * Use a Unified Shader Model – write one shader which can produce several different visual effects
 - * Group objects which are always drawn together into a display list





Performance 3

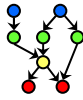
- Cull out objects which aren't visible
 - * Quick test: if the bounding box is behind the camera...
 - * Better tests will take into account field of view, distance, occlusion (can't see that room if you're in this room), etc.
- Precompute lighting for static lights and models
- Avoid round trips, like glReadBuffer
 - * Completely stalls the graphics hardware
 - * New GL extensions help you avoid them
 - ⇒ Render directly to a texture
 - ⇒ Asynchronous reads using pixel buffer objects



Performance 4

- Use texture formats supported by the hardware
 - * GL_BGRA
- Use pixel buffer objects to load textures quickly
 - * If possible, keep all your textures on the card
 - * If not, use PBOs so the card can directly read the texture data
- Measure! Measure! Measure!
 - * You never know what change you're going to make which will kick you off the fast path, or even worse, require software rendering...





Learning More

- The Red Book: The OpenGL Programming Guide
 - * A good tutorial and guide to OpenGL
- www.opengl.org
 - * The OpenGL Specification
 - * GL Extensions
- www.nvidia.org
 - * Lots of papers about how to draw pretty pictures quickly
- UPL GameSIG
 - * Come meet other students and join projects

