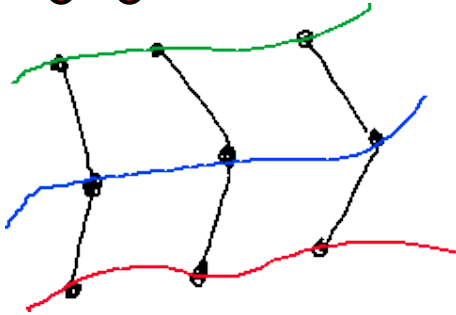


Lecture 17 of 42



Scene Graphs in OpenGL

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>

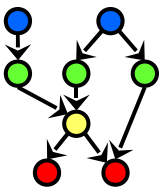
Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Readings:

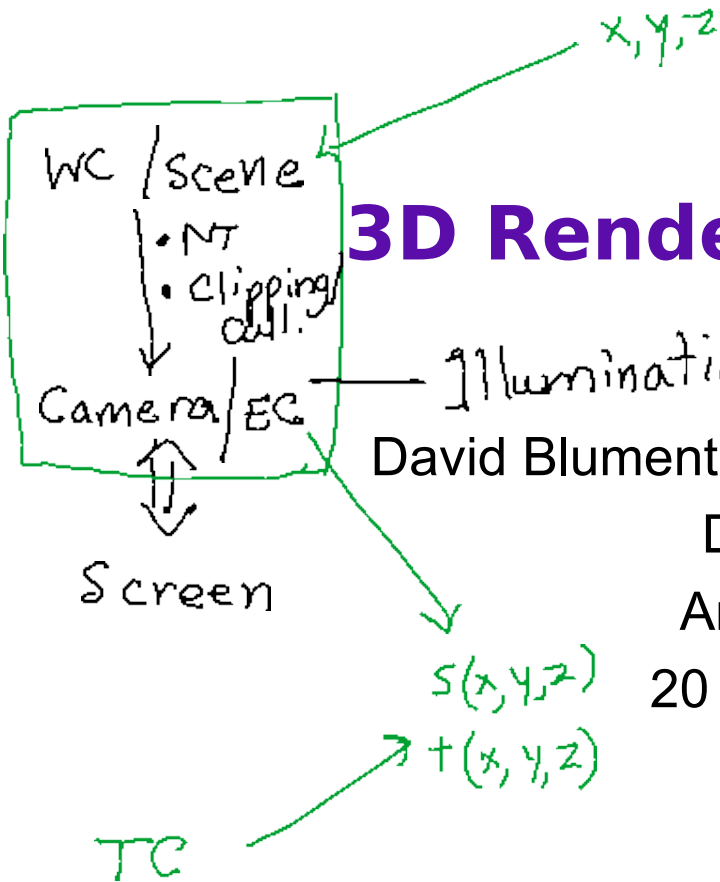
Sections 4.1 – 4.3, Eberly 2^e – see <http://snurl.com/1ye72>

Friday: Sections 4.4 – 4.6, Eberly 2^e





MC



3D Rendering in OpenGL

Illumination

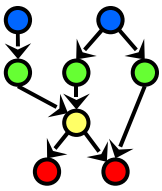
David Blumenthal, SkyJuggler Consulting

David Murray

Anthony Magro

20 February 2006

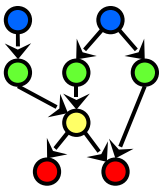




Goals for this Tutorial

- Show you a simple 3D application
 - * Discuss the basic elements of an OpenGL program
 - * Give you enough familiarity with OpenGL that you can go learn the details on your own
- Hows and Whys
 - * Concepts behind how OpenGL works
 - * Things you might not get from reading the code
- Practical Tips
 - * Code and data organization
 - * Performance

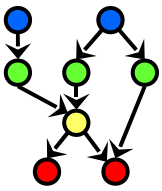




Outline

- Look at some drawing code
- Coordinate Spaces and Transformations
- Program And Data Structure
- Performance
- OpenGL in SolidWorks
- Q&A and Open Discussion

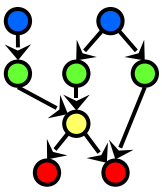




Looking at the Code

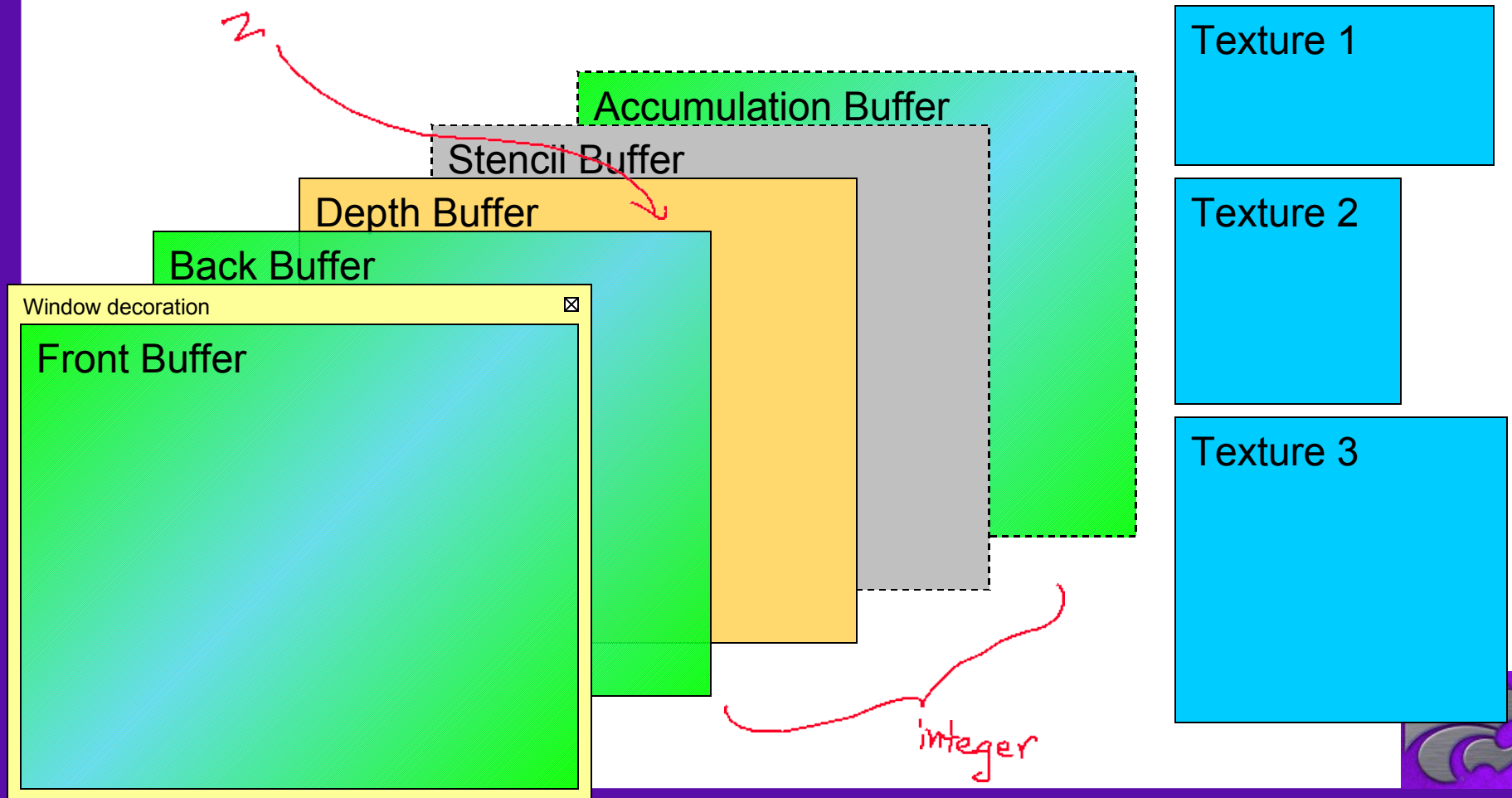
- Setup OpenGL State
- Geometry precomputation
- Frame Loop
- Drawing Code

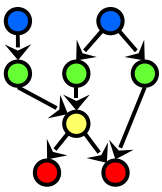




Draw Buffers

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





Why Flush?

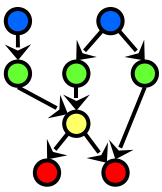
- Commands in a graphics pipeline get buffered
 - * Lowers the cost of sending data to the graphics card
 - * No guarantee the buffer ever gets sent
 - * glFlush ensures the commands will *eventually* complete
 - * Generally a SwapBuffer implies a glFlush

```
...  
glVertex  
glVertex  
glVertex  
glVertex  
...
```



DMA Buffer

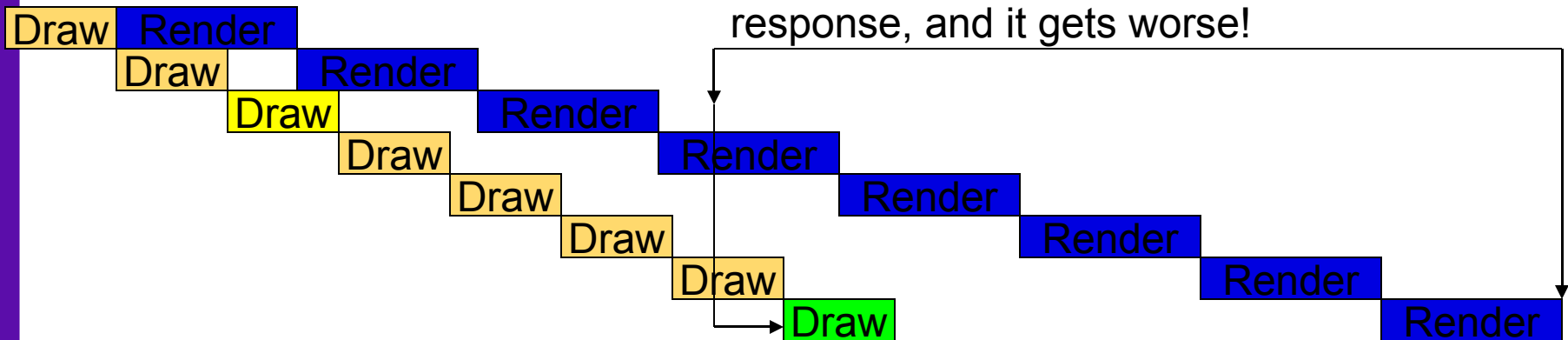




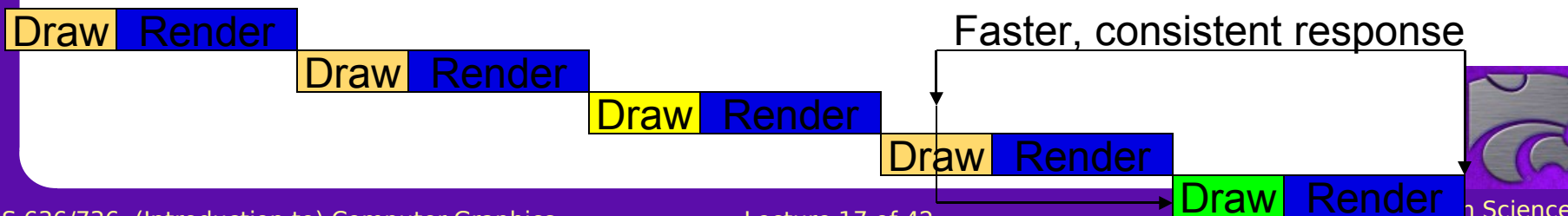
Why Finish?

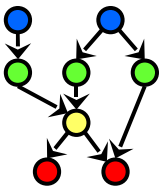
- `glFinish` = `glFlush` + wait for completion
 - * This can slow down the frame rate, but...
 - * It increases responsiveness!

Drawing as fast as possible:



Using `glFinish`:

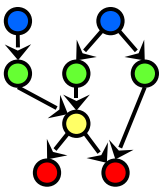




Questions

- Any questions about the code so far?

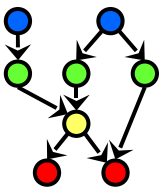




Coordinate Spaces

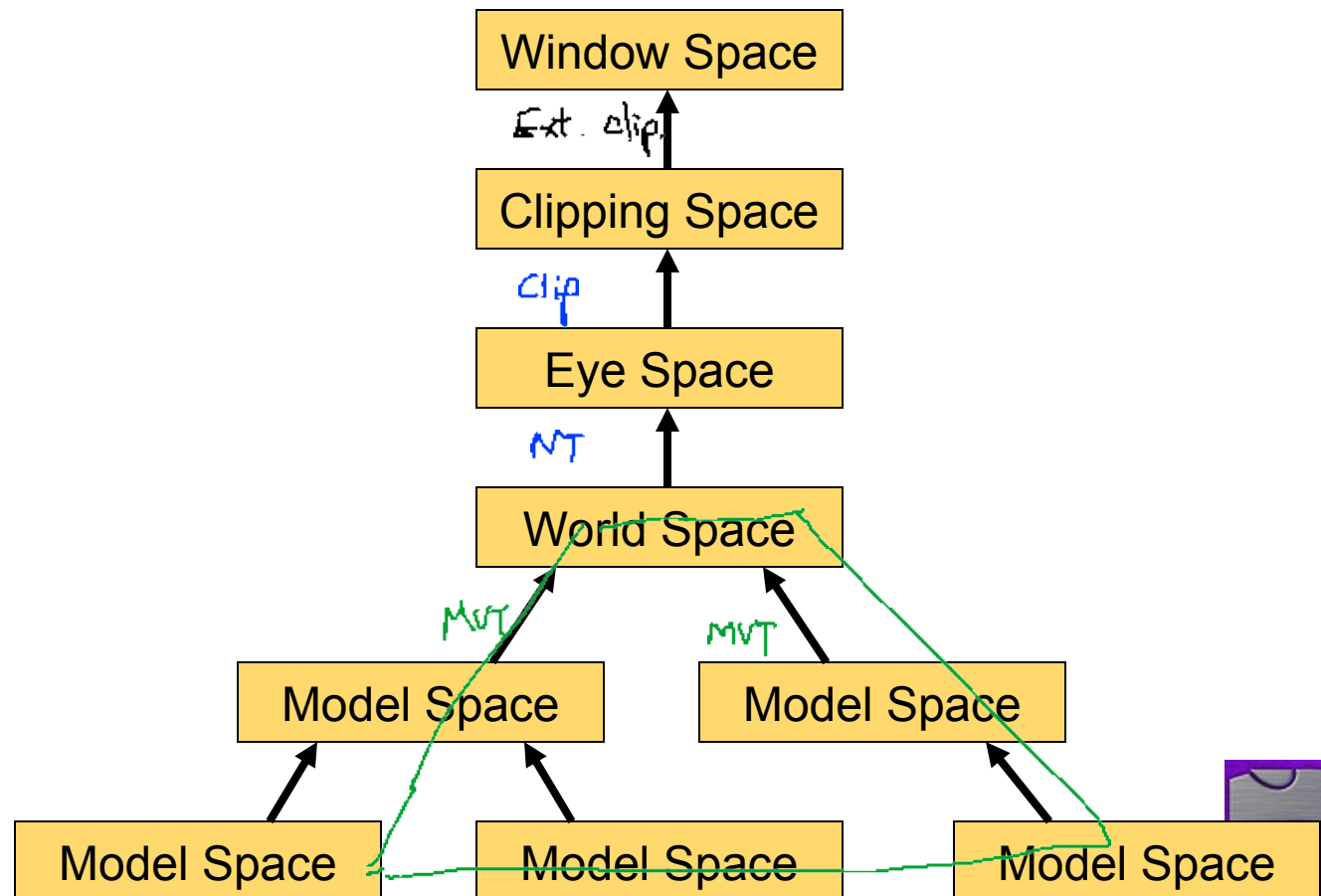
- Think of space in terms of an origin and axes
 - * From inside the space:
 - ⇒ The origin is $(0, 0, 0)$, the X axis is $(1, 0, 0)$, etc...
 - ⇒ Right Hand Rule!
- A space can be positioned inside another space
 - * A transformation describes how to move from the “child” space to the “parent” space
 - * Translate, rotate, scale, skew, invert, and project
- Use whatever spaces are useful to you
 - * My personal space → this room → the building → Madison → the world → the solar system → the galaxy → the Universe → ???

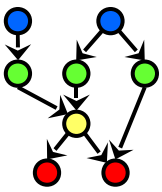




OpenGL Coordinate Spaces

- 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.

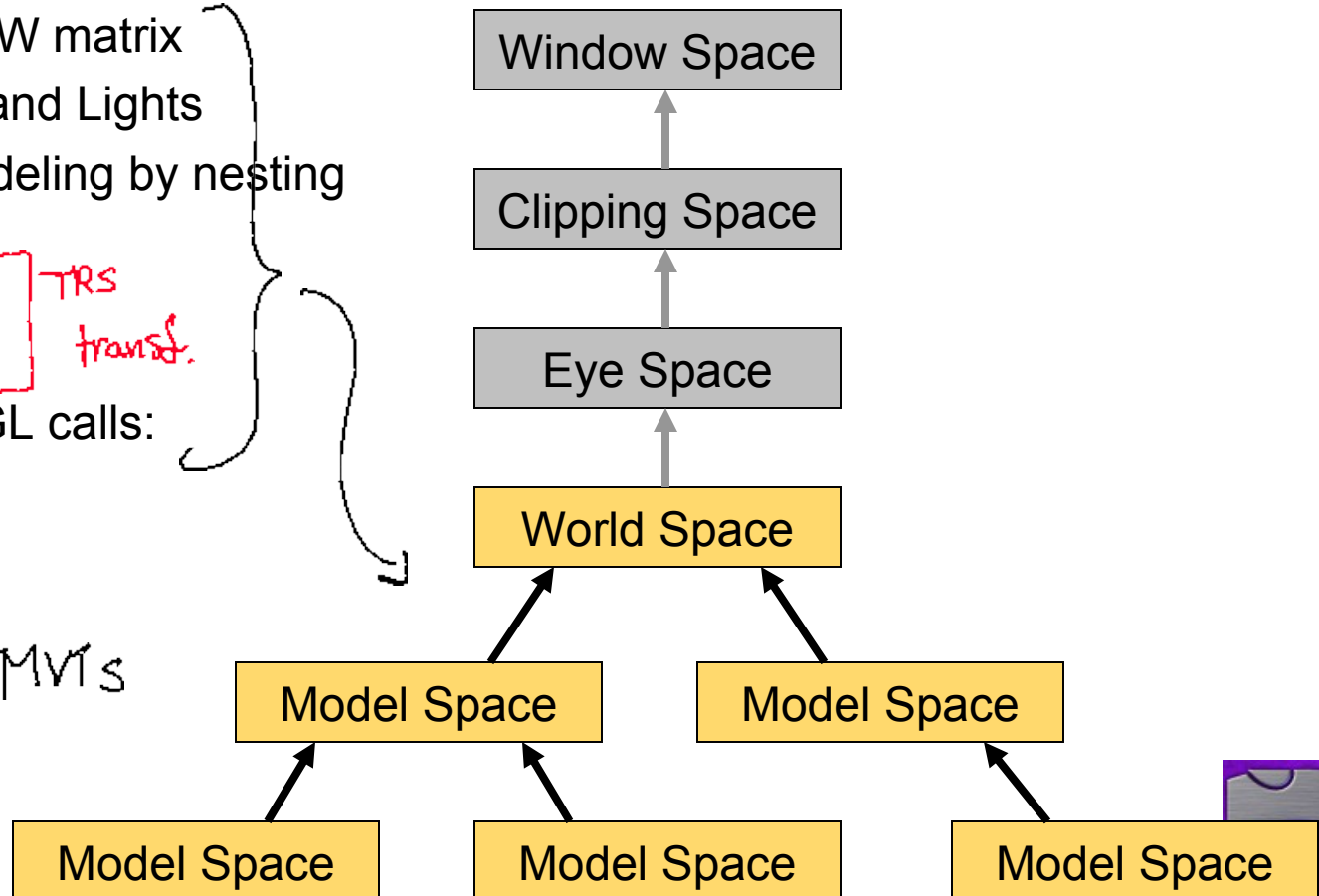
- GL_MODELVIEW matrix
- Define Objects and Lights
- Hierarchical Modeling by nesting transforms

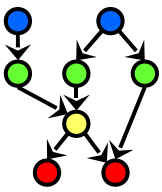
- * glPushMatrix
 - * glPopMatrix
- TRS transf.

- Most common GL calls:

- * glTranslate
- * glRotate

MVS

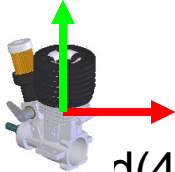




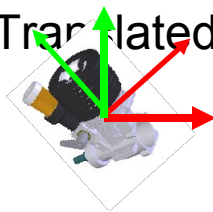
Order is Important

Non-commutative

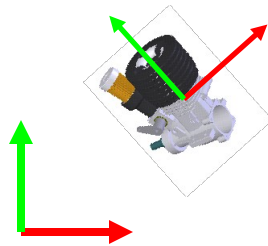
- `glLoadIdentity()`



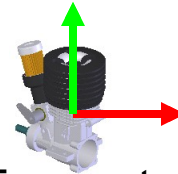
- `glTranslated(45, 0, 0)`



- `glTranslated(5, 0, 0)`

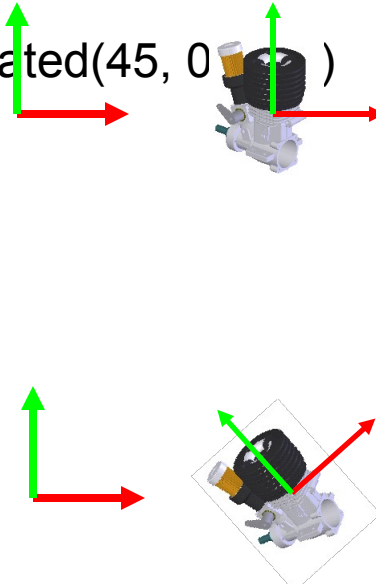


- `glLoadIdentity()`



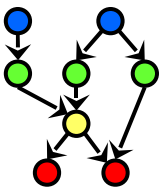
- `glTranslated(5, 0, 0)`

- `glRotated(45, 0, 0)`



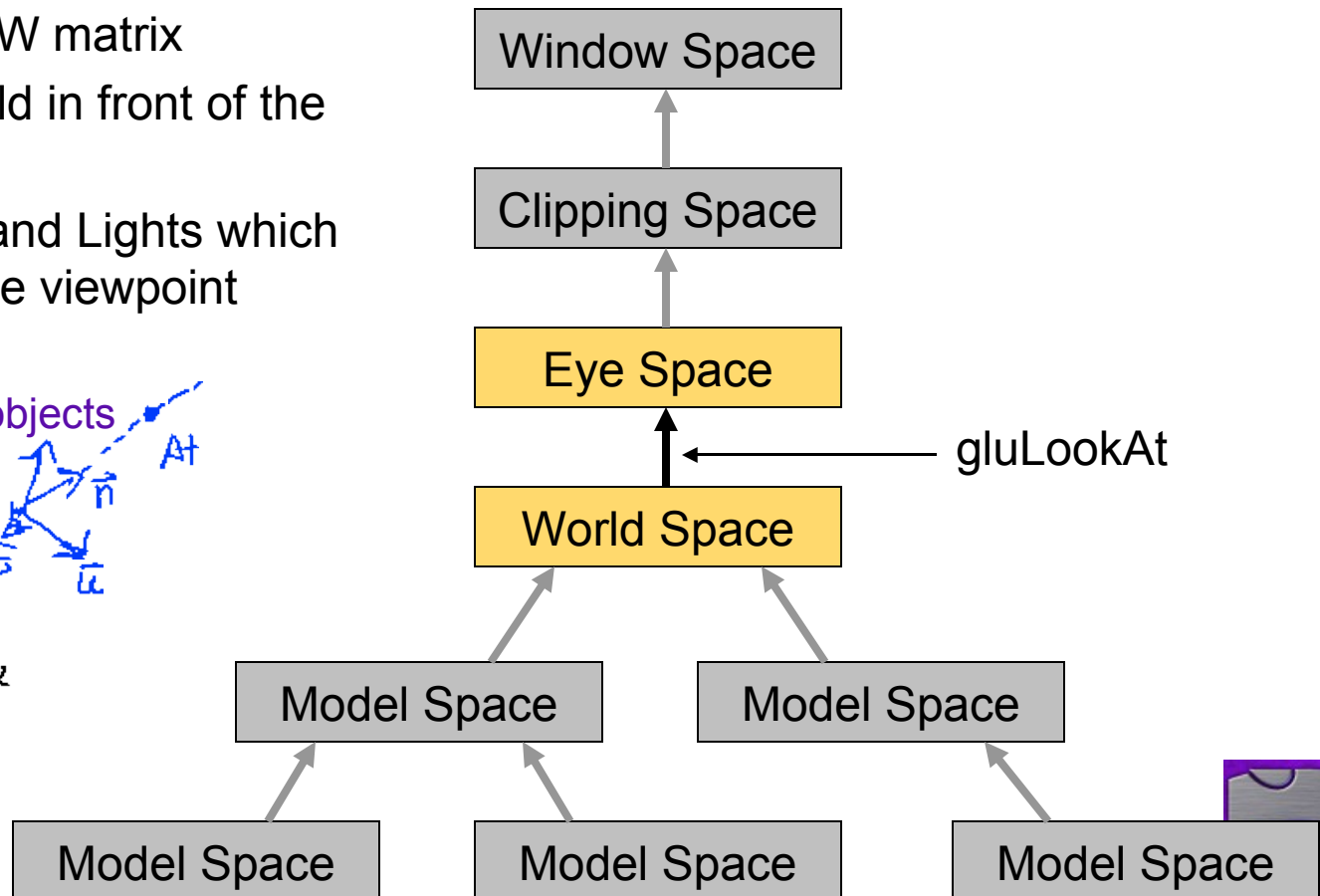
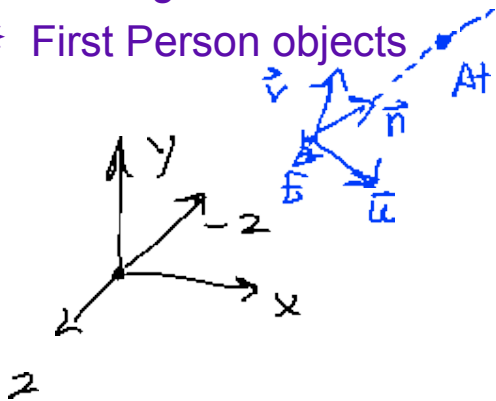
OpenGL commands successively define new “local” coordinate spaces in terms of the “current” or previous local space.

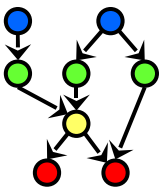




Viewing Transformation

- 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

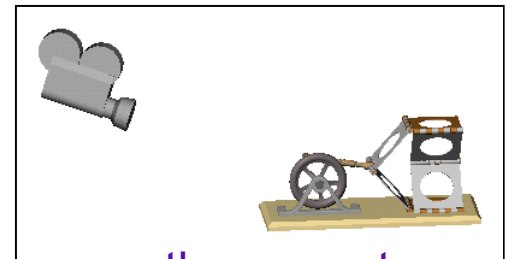




Positioning the View

- 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

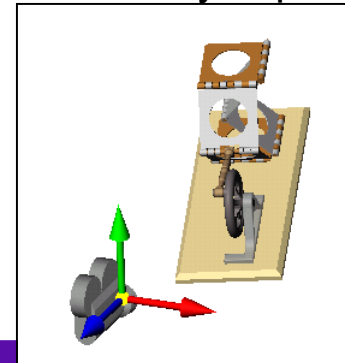
NT (hard)

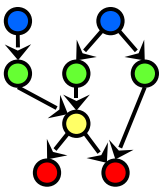


the correct
Camera in world space



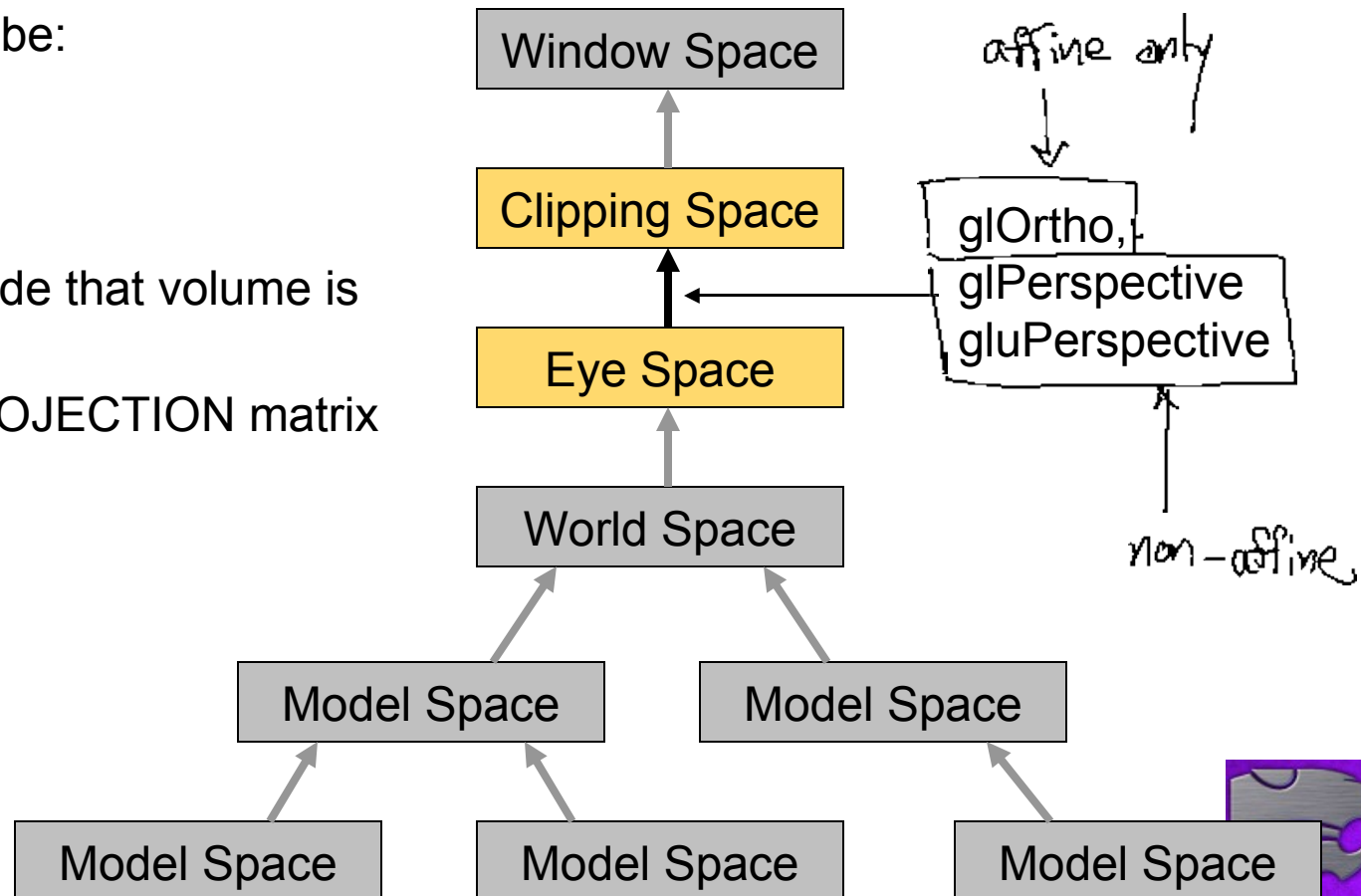
World in eye space

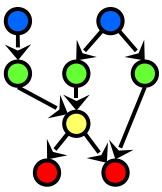




Projection Transformation

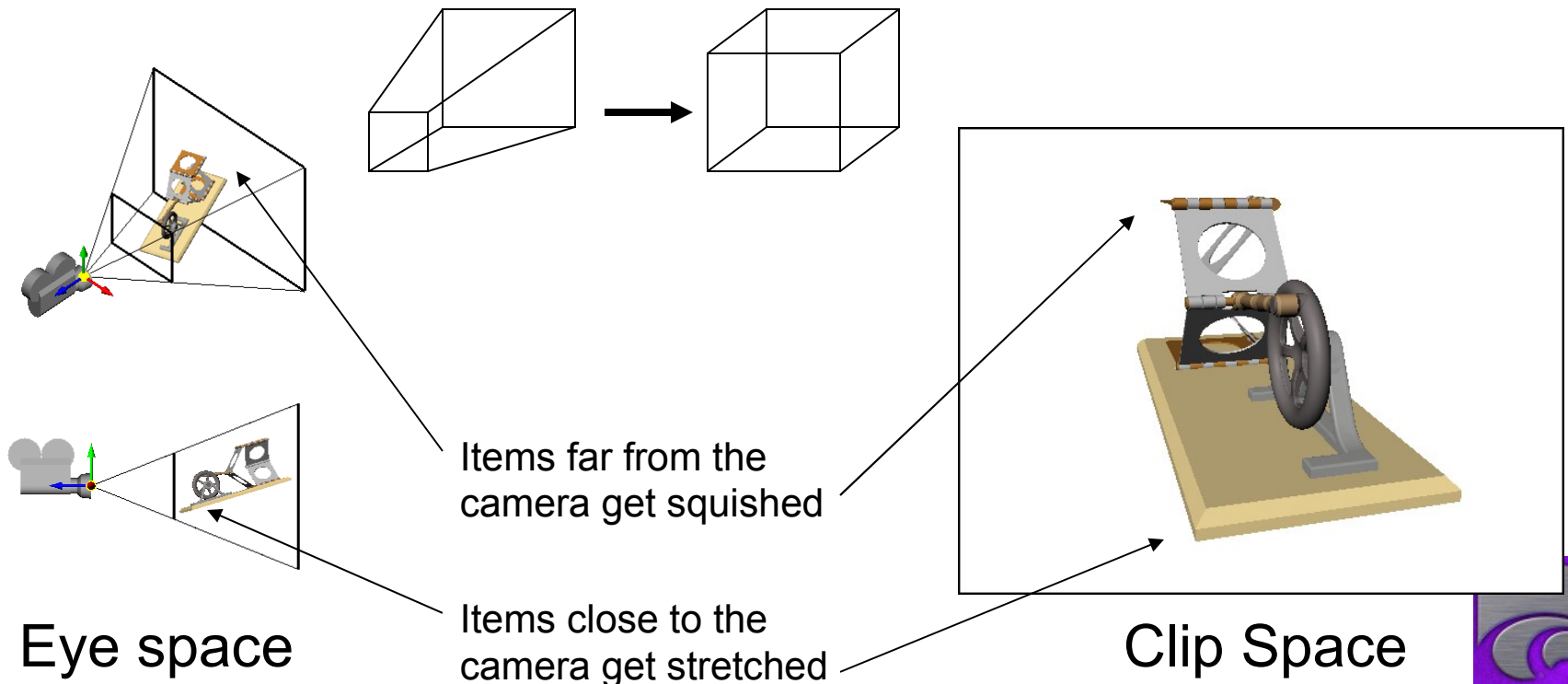
- Transform the region to be displayed into the region of the Clipping Cube.
- The Clipping Cube:
 - * -1 to 1 in X
 - * -1 to 1 in Y
 - * -1 to 1 in Z
- Everything outside that volume is clipped out.
- Use the GL_PROJECTION matrix

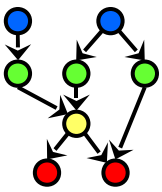




Perspective Projection

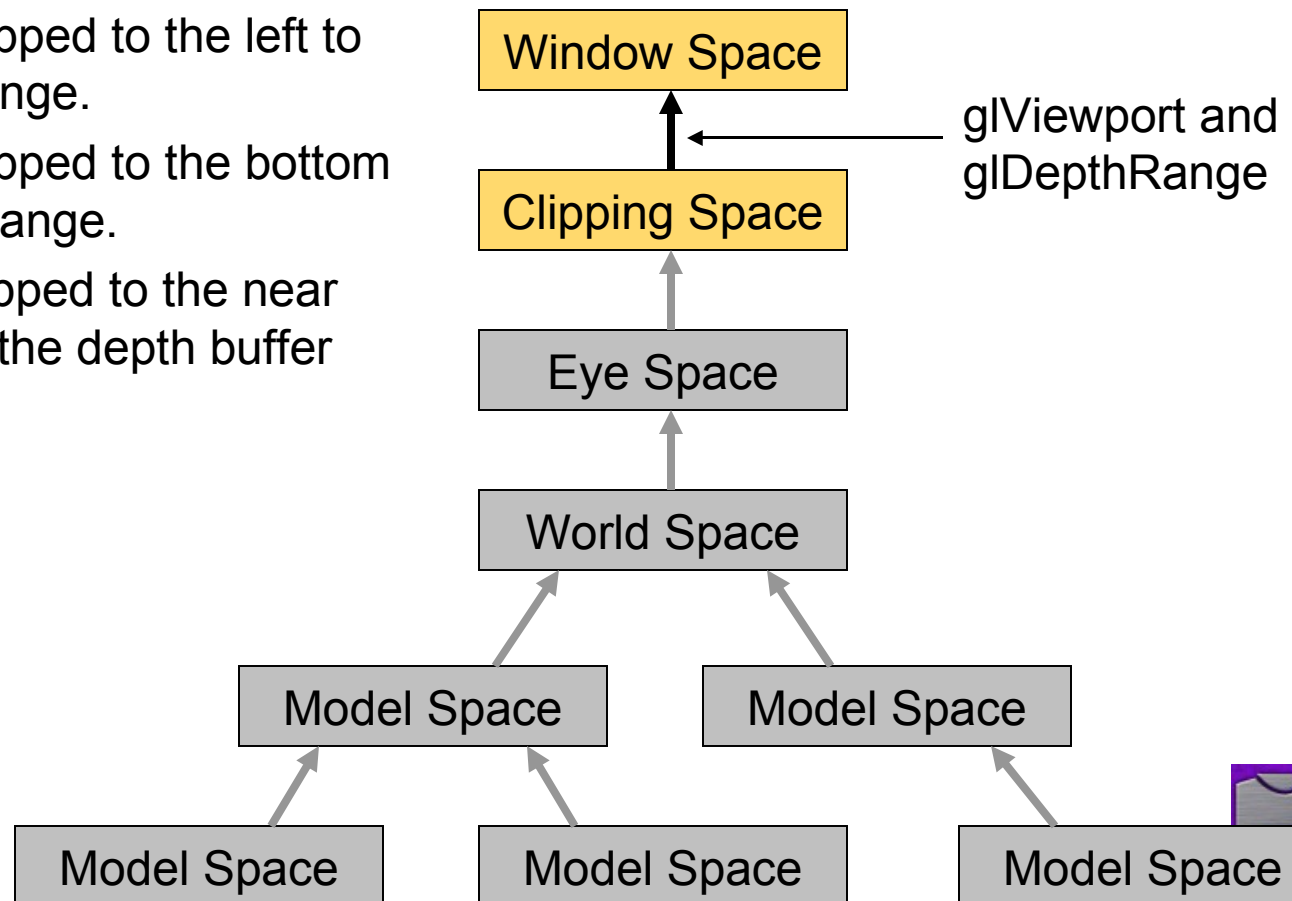
- The `GL_PROJECTION` matrix maps a region of eye space to the clipping space cube.
- `glFrustum` creates a non-linear mapping, yielding a perspective effect.

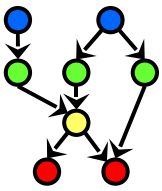




Viewport Transformation

- 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.

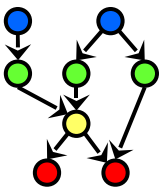




Questions

- Any questions about coordinate spaces or transformations?

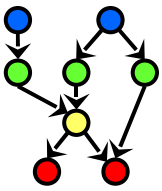




Program Structure

- Do everything that you can in the Setup function
 - * All file I/O: model, texture, and shader loading
 - * Setup OpenGL state, anything that doesn't change
 - * Prepare display lists, vertex and pixel buffer objects
 - * Setup program logic, animation controls, etc.
- Keep the Frame Loop fast
 - * Advance time, update program and animation state
 - * Minimize OpenGL state changes and draw calls
 - * Performance and Pretty Pictures!

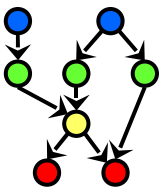




OpenGL Setup

- 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

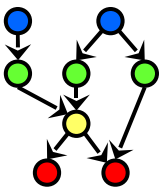




Animation Logic

- Update at the start of every frame
- Try to use elapsed realtime as much as possible
 - ✦ Avoid fixed increments per frame
 - ✦ Throttle the increment in case of chronic or transient bad performance
- Create an Event Queue and a Behavior Manager
 - ✦ Ability to schedule tasks for the future
 - ✦ Ability to wrap AIs and repetitive actions inside an easy-to-maintain structure
 - ✦ Add new events and behaviors based on program logic

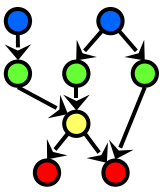




Scene Graphs

- 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

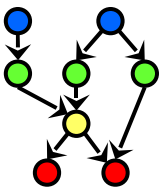




Scene Graphs (continued)

- 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

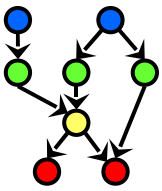




Don't Reinvent the Wheel

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

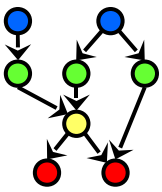




Questions

- Any questions about code and data organization?

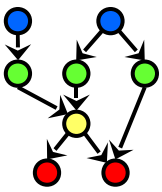




Performance

- 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

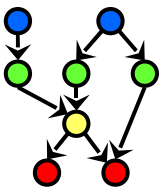




Performance 2

- 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

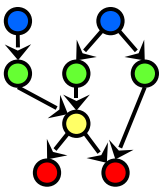




Performance 3

- 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

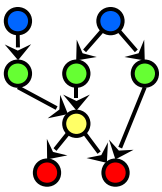




Performance 4

- 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

