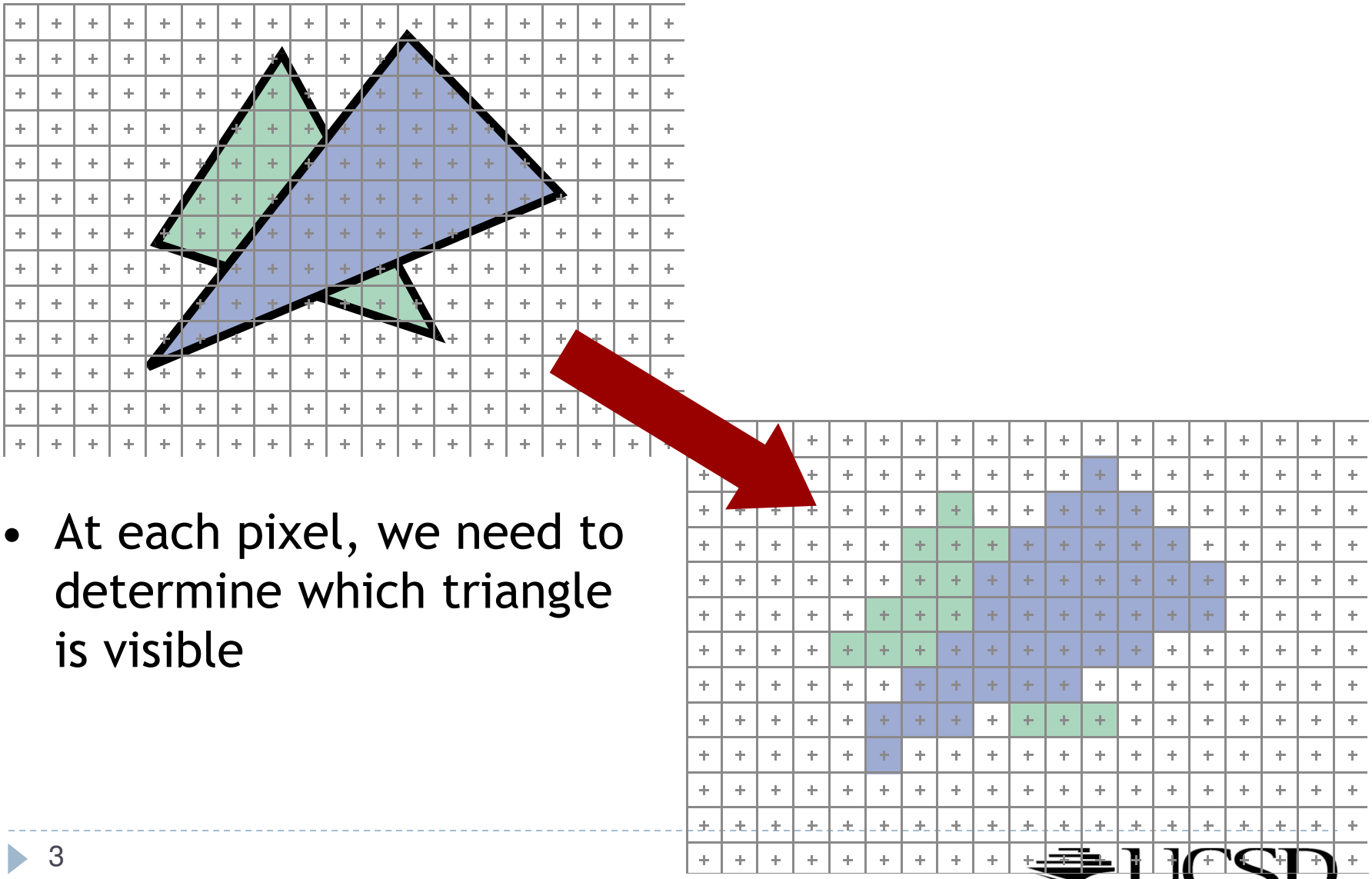
CSE 167:
Introduction to Computer Graphics
Lecture #5: Visibility, OpenGL

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2016

Announcements

- Tomorrow: assignment 1 due
- Grading starts at 2pm in labs 260 and 270
- Need to upload code to TritonEd by 2pm
- We'll be grading at least until 4pm

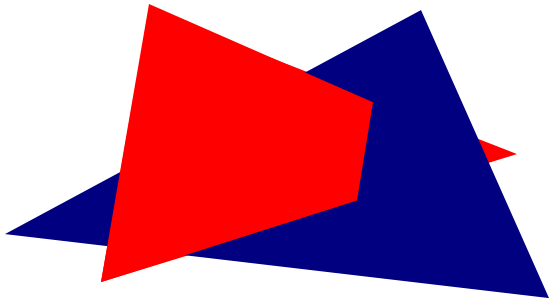
Visibility



- At each pixel, we need to determine which triangle is visible

Painter's Algorithm

- Paint from back to front
- Need to sort geometry according to depth
- Every new pixel always paints over previous pixel in frame buffer
- May need to split triangles if they intersect



- Intuitive, but outdated algorithm - created when memory was expensive
- Needed for translucent geometry even today

Z-Buffering

- Store z-value for each pixel
- Depth test
  - Initialize z-buffer with farthest z value
  - During rasterization, compare stored value to new value
  - Update pixel only if new value is smaller

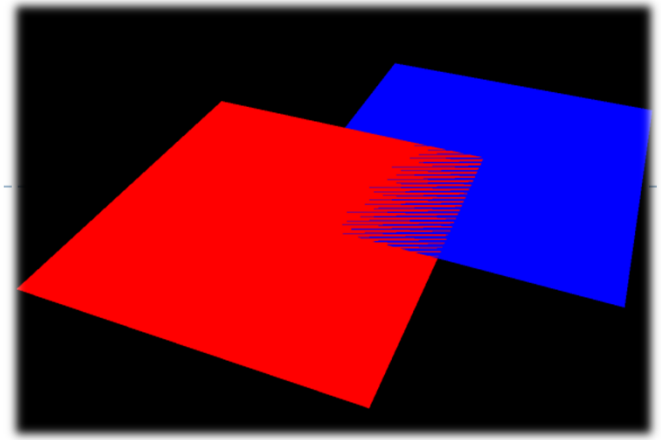
```
setpixel(int x, int y, color c, float z)
if(z<zbuffer(x,y)) then
  zbuffer(x,y) = z
  color(x,y) = c
```

- z-buffer is dedicated memory reserved in GPU memory
- Depth test is performed by GPU → very fast

Z-Buffering in OpenGL

- In OpenGL applications:
  - Ask for a depth buffer when you create your GLFW window.
    - glfwOpenWindow(512, 512, 8, 8, 8, 0, **16**, 0, GLFW_WINDOW)
  - Place a call to glEnable(GL_DEPTH_TEST) in your program's initialization routine.
  - Ensure that your *zNear* and *zFar* clipping planes are set correctly (glm::perspective(fovy, aspect, zNear, zFar)) and in a way that provides adequate depth buffer precision.
  - Pass GL_DEPTH_BUFFER_BIT as a parameter to glClear.
- Note that the z buffer is non-linear: it uses smaller depth bins in the foreground, larger ones further from the camera.

Z-Buffer Fighting



- Problem: polygons which are close together don't get rendered correctly. Errors change with camera perspective → flicker
- Cause: differently colored fragments from different polygons are being rasterized to same pixel and depth → not clear which is in front of which
- Solutions:
  - move surfaces farther apart, so that fragments rasterize into different depth bins
  - bring near and far planes closer together
  - use a higher precision depth buffer. Note that OpenGL often defaults to 16 bit even if your graphics card supports 24 bit or 32 bit depth buffers

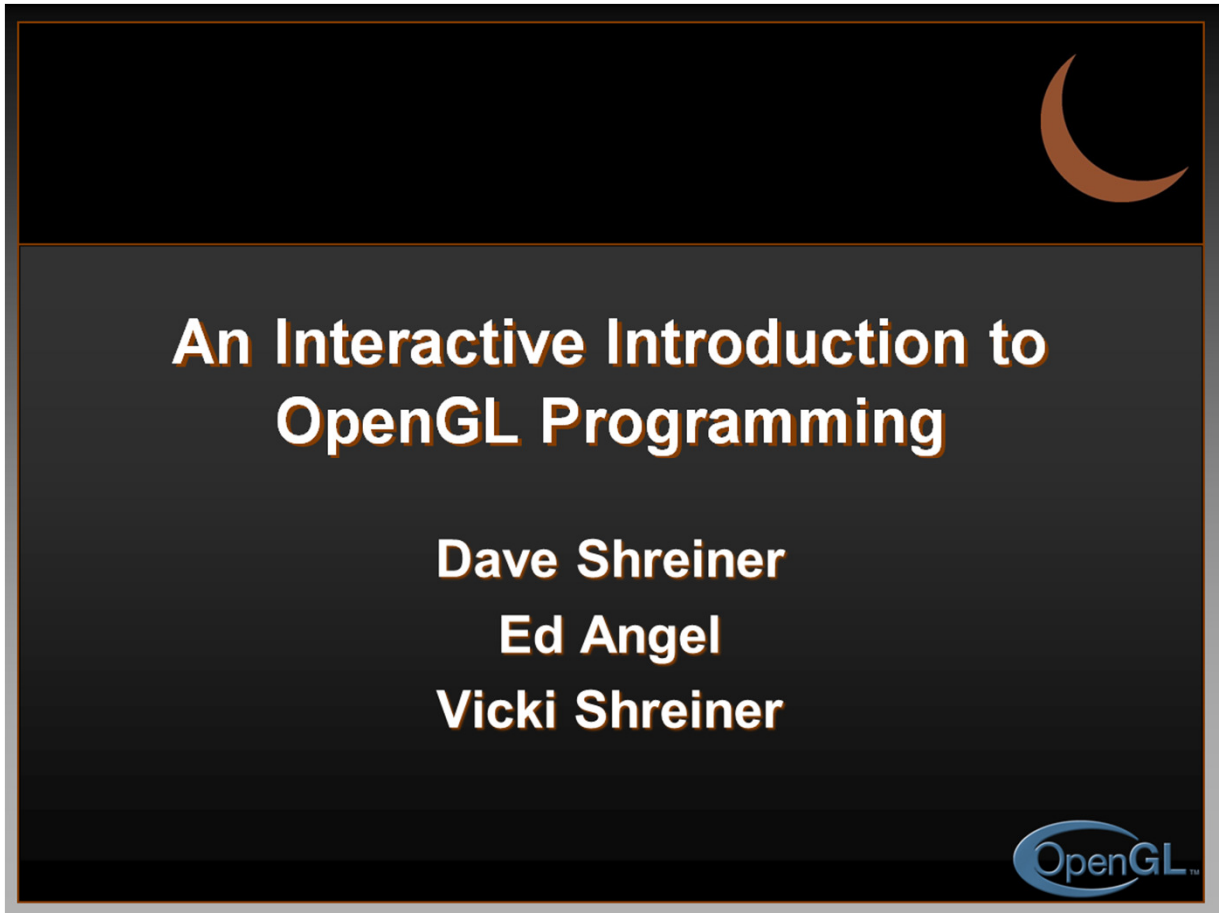
Translucent Geometry

- Need to depth sort translucent geometry and render with Painter's Algorithm (back to front)
- Problem: incorrect blending with cyclically overlapping geometry

- Solutions:
  - Back to front rendering of translucent geometry (Painter's Algorithm), after rendering opaque geometry
    - Does not always work correctly: programmer has to weigh rendering correctness against computational effort
  - Theoretically: need to store multiple depth and color values per pixel (not practical in real-time graphics)

Introduction to OpenGL

- Using slides from SIGGRAPH course:



UCSD

OpenGL and GLFW Overview

- What is OpenGL & what can it do for me?
- OpenGL in windowing systems
- Why GLFW
- A GLFW program template

What Is OpenGL?

- Graphics rendering API
  - high-quality color images composed of geometric and image primitives
  - window system independent
  - operating system independent

OpenGL as a Renderer

- Geometric primitives
  - points, lines and polygons
- Image Primitives
  - images and bitmaps
  - separate pipeline for images and geometry
    - linked through texture mapping
- Rendering depends on state
  - colors, materials, light sources, etc.

Related APIs

- GLU (OpenGL Utility Library)
  - part of OpenGL
  - NURBS, tessellators, quadric shapes, etc.
- GLFW (OpenGL Utility Toolkit)
  - portable windowing API
  - not officially part of OpenGL

Preliminaries

- Headers Files
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GLFW/glfw3.h>
- Libraries
- Enumerated Types
  - OpenGL defines numerous types for compatibility
    - GLfloat, GLint, GLenum, etc.

GLFW Basics

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Enter event processing loop

Sample Program

```
#include <GLFW/glfw3.h>

int main(void)
{
    GLFWwindow* window;

    /* Initialize the library */
    if (!glfwInit()) return -1;

    /* Create a windowed mode window and its OpenGL context */
    window = glfwCreateWindow(640, 480, "Hello CSE 167", NULL, NULL);
    if (!window)
    {
        glfwTerminate();
        return -1;
    }
    /* Make the window's context current */
    glfwMakeContextCurrent(window);

    /* Initialize OpenGL here */

    /* Loop until the user closes the window */
    while (!glfwWindowShouldClose(window))
    {
        /* Render here with OpenGL */

        /* Swap front and back buffers */
        glfwSwapBuffers(window);

        /* Poll for and process events */
        glfwPollEvents();
    }
    glfwTerminate();
    return 0;
}
```

OpenGL Initialization

- Set up whatever state you are going to use

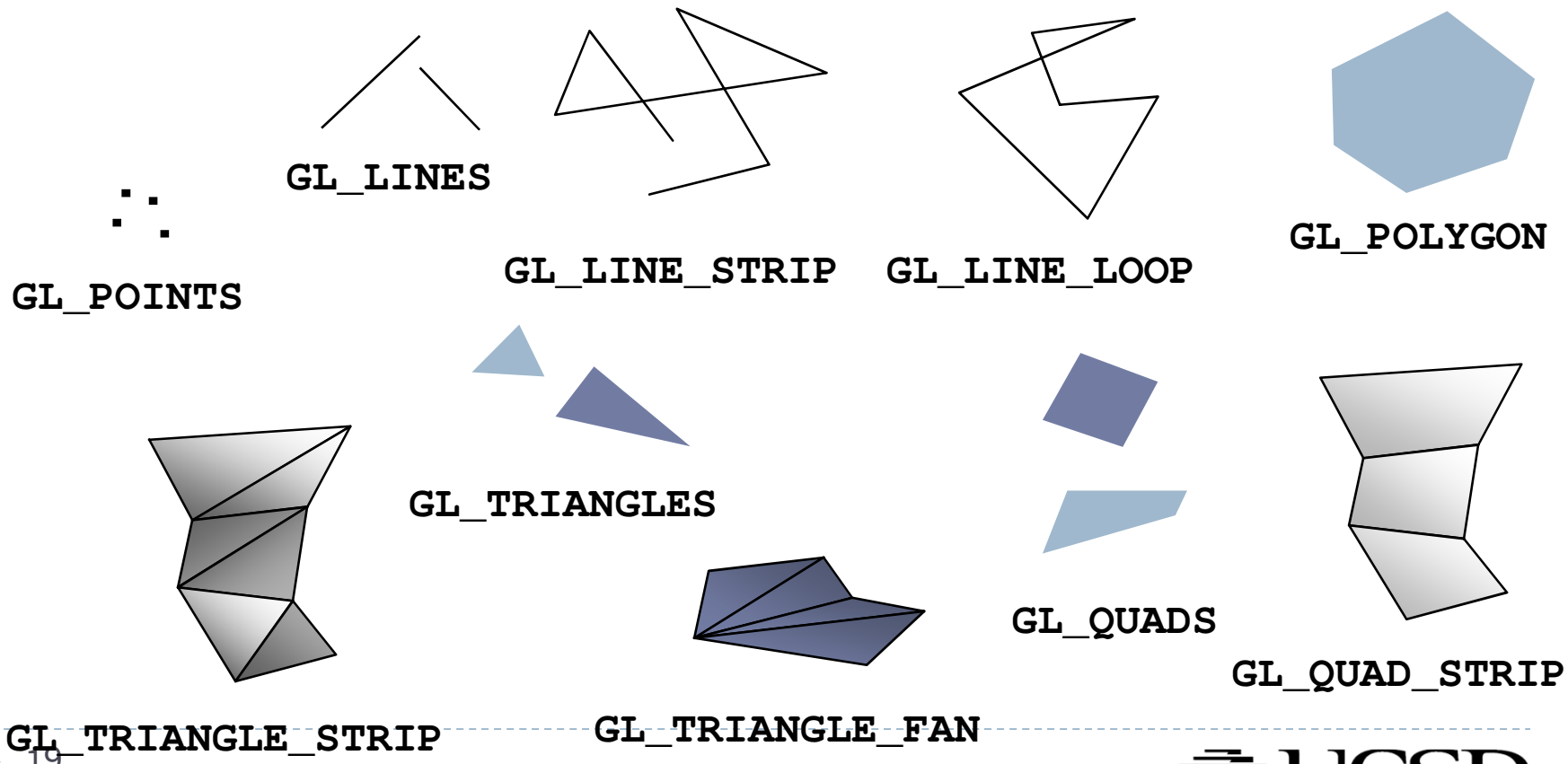
```
void init( void )
{
  glClearColor( 0.0, 0.0, 0.0, 1.0 );
  glClearDepth( 1.0 );

  glEnable( GL_LIGHT0 );
  glEnable( GL_LIGHTING );
  glEnable( GL_DEPTH_TEST );
}
```

Elementary Rendering

- Geometric Primitives
- Managing OpenGL State
- OpenGL Buffers

OpenGL Geometric Primitives

- All geometric primitives are specified by vertices



UCSD

OpenGL's State Machine

- All rendering attributes are encapsulated in the OpenGL State
  - rendering styles
  - texture mapping
  - control of programmable shaders

Manipulating the OpenGL State

- Appearance is controlled by current state

```
for each ( primitive to render )
{
    update OpenGL state
    render primitive
}
```

Manipulating the OpenGL State

- Setting the State

```
glPointSize( size );
glLineStipple( repeat, pattern );
```

- Enabling Features

```
glEnable( GL_LIGHTING );
glDisable( GL_TEXTURE_2D );
```

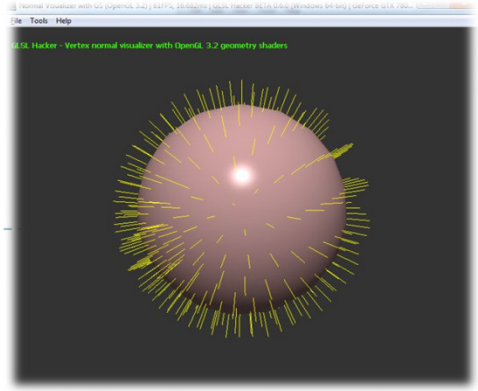
Debugging OpenGL Code

OpenGL error state: glGetError()

- OpenGL has an error state
- Use glGetError() to find location of error. It will clear the error flag.
- Then gluErrorString() to parse the error message

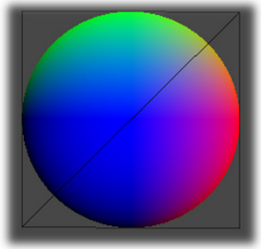
```
void printGLError(const char* msg)
{
  const GLenum err = glGetError();
  if(err != GL_NO_ERROR)
  {
    const char* str = (const char*)gluErrorString(err);
    cerr << “OpenGL error: " << msg << ", " << str << endl;
  }
}
```

Tips for Visual Debugging



Normal Visualizer

- Collisions, view frustum culling:
  - Show bounding boxes/spheres for all objects
- Problems with shading:
  - Display normal vectors on vertices as line segments pointing in the direction of the vector. Example: Normal Visualizer (pictured above).
  - Or interpret surface normals as RGB colors by shifting x/y/z range from -1..1 to 0..1.
- Display direction and other vectors:
  - Display normal vectors as described above.
- Objects don't get rendered:
  - Find out if they won't render or are just off screen by temporarily overwriting GL_MODELVIEW and GL_PROJECTION matrices with simpler ones, and/or zooming out by increasing the field of view angle.



Normal shading

OpenGL Debugging Tools

- Overview with many links:
  - https://www.opengl.org/wiki/Debugging_Tools
- Nvidia tools (Nsight and others):
  - https://developer.nvidia.com/gameworks-tools-overview