CSCI 420 Computer Graphics

Helper slides, hw1 (height field)

Jernej Barbic
University of Southern California

Important first steps

- There must be glutSwapBuffers() at the end of displayFunc()

- There must be glutPostRedisplay() at the end of idleFunc()

Understanding modelview and projection matrices

- 4x4 matrices
- You compute them using the OpenGLMatrix class
- You send them to the shader using glUniformMatrix4fv

- There are two OpenGLMatrix modes: ModelView and Projection
- Use OpenGLMatrix::SetMatrixMode to set the mode

Computing the projection matrix

- Compute in reshape()
- Change the mode to Projection
- Clear the matrix to identity (OpenGLMatrix::LoadIdentity)
- Then, call OpenGLMatrix::Perspective
- Good habit to then set the mode back to ModelView

Uploading the projection matrix to GPU

Inside displayFunc():

```
float p[16];
openGLMatrix->SetMatrixMode(OpenGLMatrix::Projection);
openGLMatrix->GetMatrix(p);
```

- Then, upload the array p to the GPU:
  See the "Setting up uniform variables" slides
  in the "04-Shaders" lecture.

Computing the modelview matrix

- Compute in displayFunc()
- Change the mode to ModelView
  openGLMatrix->SetMatrixMode(OpenGLMatrix::ModelView);
- Clear the matrix to identity (OpenGLMatrix::LoadIdentity)
- Then, call OpenGLMatrix::LookAt()
- Then, call OpenGLMatrix::Translate,Rotate,Scale
- Then

```
float m[16];
openGLMatrix->GetMatrix(m);
```

- Then, upload the array m to the GPU

Initialization

- Init and bind the pipeline program:
  pipelineProgram->Init("../openGLHelper-starterCode");
  pipelineProgram->Bind();

- Generate the VBO and VAO, and properly upload them to the GPU

  See the "Vertex Array Object" slides ("04-Shaders" lecture), and the "Vertex Buffer Object" slides ("03-Interaction" lecture).

Write the vertex and fragment shaders

- See “04-Shaders”:
  “Basic Vertex Shader in GLSL” and
  “Basic Fragment Shader”

Note: shaders are already fully written in the starter code.

Heightfield VBOs and VAOs

- 1 VBO + 1 VAO for solid mode
  1 VBO + 1 VAO for wireframe mode
  1 VBO + 1 VAO for point mode

- VBO contains positions and colors

- Others designs are OK too (separate VBOs for positions and colors)

Gotchas to avoid

- First, initialize OpenGL.
- VAO must be initialized AFTER the pipeline program has been initialized and bound.
- VAO must be initialized AFTER setting up VBO.
- The order of setting up the VBO and the pipeline program does not matter.

- Data sent to VBO must be contiguous.

```
float* vertices[36];
vertices[0] = new float[3];
vertices[1] = new float[3];
…
```

Rendering (in displayFunc)

- Setup modelview and projection matrices
  (as shown in the previous slides in this presentation)
- Bind the VAO
- Render using glDrawArrays()
- See "Use the VAO" slide in "04-Shaders"

Complete sequence
of steps to render
1 triangle

1. Set up the basics

Temporarily change glClearColor to:

```
glClearColor(0.0f, 1.0f, 0.0f, 0.0f);
```

(so that we can see a green background)

During initialization, enable hidden surface removal:

```
glEnable(GL_DEPTH_TEST);
```

In displayFunc(), clear the color and depth buffers:

```
glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT);
```

At the end of displayFunc(), swap the buffers:

```
glutSwapBuffers();
```

You should now see a green screen.

2. Create geometry (one triangle)

During initialization:

```
float positions[] = { 0, 0, -1,
                      1, 0, -1,
                      0, 1, -1};
float colors[] = { 1, 0, 0, 1,
                   0, 1, 0, 1,
                   0, 0, 1, 1};
```

3. Create VBO for it

See lecture “Interaction”,
slide “Vertex Buffer Object: Initialization"

4. Initialize openGLMatrix and pipelineProgram

1. openGLHelper:

Make a global variable:

OpenGLMatrix openGLMatrix;

2. pipeline program:

See Shaders Lecture, slide “Setting up the Pipeline Program”

5. Create handles for modelview and projection matrices

Global variables:

```
GLint h_modelViewMatrix, h_projectionMatrix;
```

During initialization:

```
GLuint program = pipelineProgram.GetProgramHandle();
h_modelViewMatrix =
  glGetUniformLocation(program, “modelViewMatrix”);
h_projectionMatrix =
  glGetUniformLocation(program, “projectionMatrix”);
```

6. Prepare the projection matrix

Do it in reshape().

See lecture “Viewing”, slide “OpenGL code (reshape)”.

7. Prepare the modelview matrix

Do it in displayFunc().

See lecture “Viewing”, slide “OpenGL code (camera positioning)”.

For good triangle viewing, position the camera at (0, 0, zStudent) (see homework instructions for zStudent).

For up vector, pick some vector orthogonal to the camera viewing direction. For example, (1, 0, 0).

8. Upload modelview and projection matrices to the shader

Do it in displayFunc().

See lecture “Viewing”, slide “OpenGL code (camera positioning)”.

9. Make the VAO

See “Shaders” lecture, slides:

- VAO code (“position” shader variable)
- VAO code (“color” shader variable)

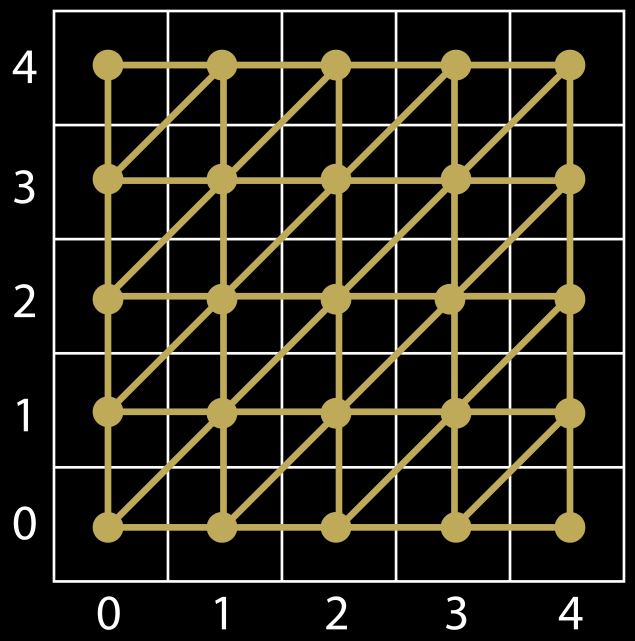
10. Use the VAO

See “Shaders” lecture, slide “Use the VAO”.

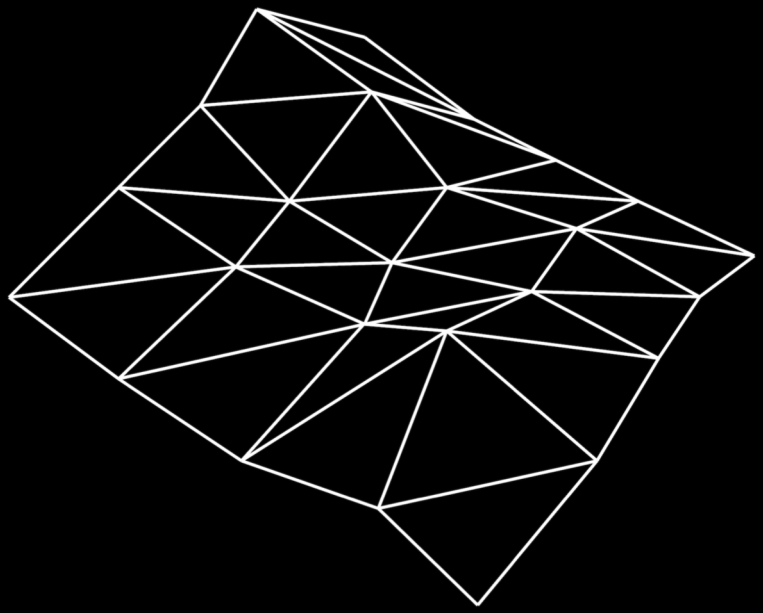
You should now see the triangle on the screen.

Rendering the height field

Understanding the Height Field



The image (5x5)

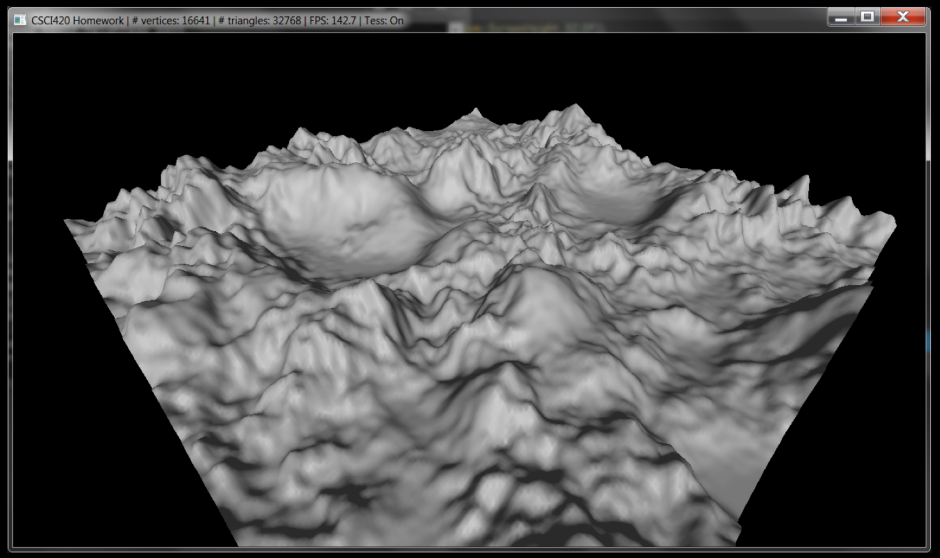


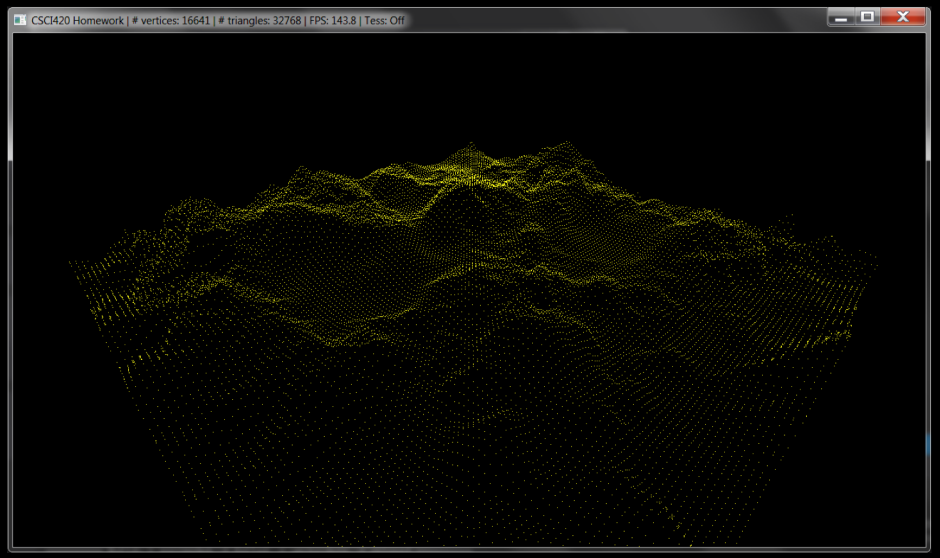
The heightfield

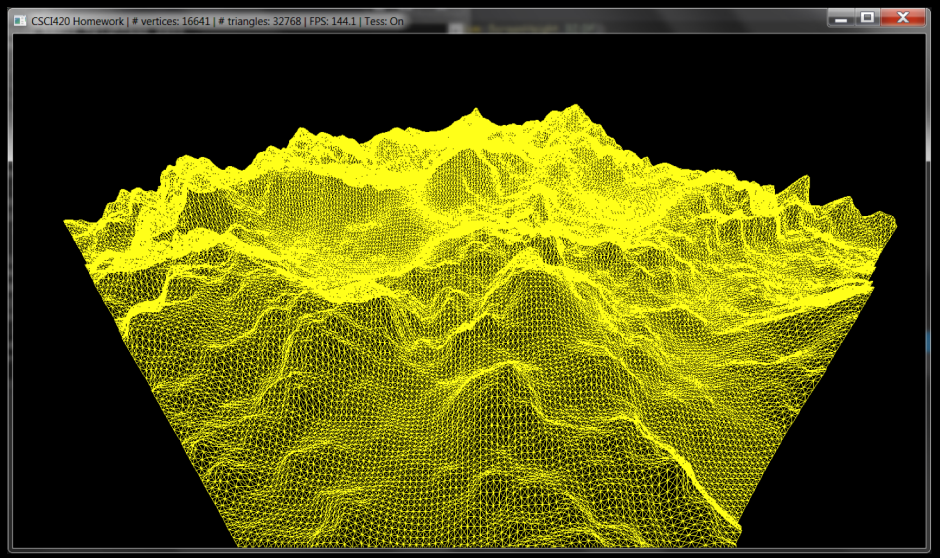
Pixel (i,j) → Vertex (i, height, -j)

```
height = scale * heightmapImage->getPixel(i, j, 0);
```

Solid, wireframe, point mode







FIRST, render a single triangle

- Do not attempt to render a heightfield until this works! ☺

- Please read the assignment description (in detail)

- MUST use the OpenGL core profile
  Do not use glBegin(), glEnd(), glVertex3f, etc.