CSCI 420 Computer Graphics

Helper slides, hw1 (height field)

Jernej Barbic
University of Southern California
Understanding the Height Field

Pixel \((i,j)\)  \rightarrow\text{Vertex} \((i, \text{height}, -j)\)

\[
\text{height} = \text{scale} \times \text{heightmapImage->getPixel}(i, j, 0);
\]
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.
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():

```c
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”
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)
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"