Line Rendering using GPU

Gen Nishida
Many Types of Shadings

• Implemented many types of shadings for our SIGGRAPH paper

Phong shading  Ambient occlusion  Line rendering  Sketchy rendering
Line Rendering

• Two-pass rendering pipeline (screen-space post-processing)
• First pass computes the normal vector and depth for each pixel and store them in texture buffers
• Second pass detects discontinuities in the normal vector and depth and defines the color accordingly
Tricky Case: Two Close Parallel Planes

- Two planes have the same normal vector and very close depth
- Simple thresholding to detect the discontinuity does not work
2D Texture

- 2D texture data can be used to store any 2d data, such as normal vector and depth

```c
glGenTextures(1, &tex);
glBindTexture(GL_TEXTURE_2D, tex);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB16F, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
```
Frame Buffer Object (FBO)

• FBO is a off-screen frame buffer. Using FBO, you can do some post-processing or rendering to a texture
• 2d texture can be attached to FBO through which the shader can store the output to the 2d texture
• For the depth, use GL_DEPTH_ATTACHMENT

```c
glGenFramebuffers(1, &fb);
glBindFramebuffer(GL_FRAMEBUFFER, fb);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texNormal, 0);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, texDepth, 0);
GLenum DrawBuffers[1] = { GL_COLOR_ATTACHMENT0 };
glDrawBuffers(1, DrawBuffers)
glEnable(GL_DEPTH_TEST);
```
1st Pass – Fragment Shader

• Store the normal vector and depth to the 2d textures that you created
• The depth is stored implicitly

```glsl
in vec3 vNormal;
layout(location = 0)out vec3 out_normal;
void main() {
    out_normal = vNormal;
    ...
}
```
Setup Textures for 2nd Pass

- Bind 0 to render to the screen
- Bind the textures where you stored the normal vector and depth for the 2nd pass

```gl
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glClearColor(1, 1, 1, 1);
glClear(GL_COLOR_BUFFER_BIT);
glDisable(GL_DEPTH_TEST);
glDepthFunc(GL_ALWAYS);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texNormal);
glActiveTexture(GL_TEXTURE2);
glBindTexture(GL_TEXTURE_2D, texDepth);
```
2nd Pass – Fragment Shader

• Obtain the normal vector and depth for each pixel
• Check if the difference in the normal and depth from the surrounding pixels is significant and set the color accordingly

```cpp
uniform sampler2D tex1; // normal
uniform sampler2D tex2; // depth

void main() {
    vec3 normal = texture(tex1, coord).xyz;
    float depth = texture(tex2, coord).x;
    ...
}
```
Sketchy Rendering

• More challenging to implement
• Convert each edge to a rectangle in the geometry shader and move it slightly closer to the camera
• Apply a texture of a sketchy stroke to the rectangle
• Better to use the screen space coordinates to compute the texture coordinates
• Use a stencil texture to make the background of the texture transparent