|  |
| --- |
| CPI 411 Graphics for Games |
| **Edge Recognition**  The main objective of this lab is to create a shader that can correctly recognize edges formed by objects in 3D space.  **A. First Concept**  Edge recognition can be performed by finding notable differences in surface normals and depth values within a small area around a given pixel. This is performed by generating a 2D map of the current camera view containing normal and depth data, then sampling adjacent pixels within that map to find discrepancies that indicate an edge.  **B. Shader File**  Fortunately, the DepthAndNormal Shader from Lab 12 perfectly serves the purpose of generating the requisite 2D map, so it can be imported as-is. The edge recognition itself requires a separate shader, named **EdgeShader.fx**. It contains the texture created by DepthAndNormal as well as a sampler to go along with it, three floats to be passed, and a vertex shader output struct containing five float2 TEXCOORDS (as well as a POSITION, which is not used but is required as input for a pixel shader to compile).  texture DepthAndNormalTexture;  float offset;  float depthDiff;  float normalDiff;  sampler normalMap = sampler\_state  {  Texture = <DepthAndNormalTexture>;  MipFilter = NONE;  MinFilter = POINT;  MagFilter = POINT;  };  struct VS\_OUTPUT  {  Float4 Position: POSITION;  float2 TexCoord: TEXCOORD0;  float2 PlusPlus: TEXCOORD1;  float2 PlusMinus: TEXCOORD2;  float2 MinusPlus: TEXCOORD3;  float2 MinusMinus: TEXCOORD4;  };  The vertex shader takes a POSITION and TEXCOORD as input and returns the VS\_OUTPUT structure. TexCoord is the pixel in question, while the other four are offsets of TexCoord in four directions. These directions are diagonal, but orthogonal directions work equally as well.  VS\_OUTPUT RenderSceneVS(float4 vPos: POSITION, float2 TexCoord : TEXCOORD0)  {  VS\_OUTPUT Output;  //step 1: find the central TEXCOORD:  Output.Position = vPos;  vPos.xy = sign(vPos.xy);  Output.TexCoord.x = (vPos.x + 1.0f) \* 0.5f;  Output.TexCoord.y = 1.0f - (vPos.y + 1.0f) \* 0.5f;  //step 2: offset in four directions:  Output.PlusPlus.x = Output.TexCoord.x+offset;  Output.PlusPlus.y = Output.TexCoord.y+offset;  Output.PlusMinus.x = Output.TexCoord.x+offset;  Output.PlusMinus.y = Output.TexCoord.y-offset;  Output.MinusPlus.x = Output.TexCoord.x-offset;  Output.MinusPlus.y = Output.TexCoord.y+offset;  Output.MinusMinus.x = Output.TexCoord.x-offset;  Output.MinusMinus.y = Output.TexCoord.y-offset;  return Output;  }    The Pixel shader compares each of these offsets with the central TexCoord. If the difference is greater than the stated threshold for either depth or normal differences, it draws the pixel as white. Otherwise, the pixel is black.  float4 RenderScenePS0(VS\_OUTPUT Input) :COLOR0  {  float4 thisPixel = tex2D(normalMap,Input.TexCoord);  float4 nextPixel = tex2D(normalMap,Input.PlusPlus);  float3 thisNormal = thisPixel.xyz \* 2.0f - 1.0f;  float3 nextNormal = nextPixel.xyz \* 2.0f - 1.0f;  if(abs(thisPixel.a - nextPixel.a) >= depthDiff || abs(thisNormal - nextNormal).x >= normalDiff || abs(thisNormal - nextNormal).y >= normalDiff || abs(thisNormal - nextNormal).z >= normalDiff)  {  return float4(1,1,1,1);  }  nextPixel = tex2D(normalMap,Input.PlusMinus);  … //repeats for each offset  return float4(0,0,0,1);  }  **C. Main Program (Game1.cs)**  This is nearly identical to Lab 12’s main program, so that should be used a baseline. The main difference is the removal of the randomNormalMap (and all statements relating to it) and changing all references to SSAO.fx to EdgeShader.fx, with necessary parameter passing.    //new variables  bool drawEdges;  float offset = .003f, depthDiff = .00005f, normalDiff = .8f;  //replaces DrawSSAO()  private void DrawEdge()  {  effect = Content.Load<Effect>("EdgeShader");  effect.CurrentTechnique = effect.Techniques[0];  effect.Parameters["DepthAndNormalTexture"].SetValue(depthAndNormalMap);  effect.Parameters["offset"].SetValue(offset);  effect.Parameters["depthDiff"].SetValue(depthDiff);  effect.Parameters["normalDiff"].SetValue(normalDiff);  effect.CurrentTechnique.Passes[0].Apply();  GraphicsDevice.DrawUserPrimitives<VertexPositionTexture>(PrimitiveType.TriangleList, vertices, 0, vertices.Length / 3);  }  **D. Main Exercise**  Import an object and observe it in the scene. Any object will work, but objects.fbx provides the most comprehensive example.    Implement key controls for changing the offset, depthDiff, and normalDiff. Play around with it and try to produce the following images:  A picture containing logo  Description automatically generated  A picture containing logo  Description automatically generated  A picture containing diagram  Description automatically generated  **\*\*\* IMPORTANT \*\*\***  Complete the exercise in D section, and submit a zipped file including the solution (.sln) file and the project folders to course online site. The submission item is located in the "**Quiz and Lab**" section. Each lab has **10 points**. If you complete the exercise in class time, the full points will be assigned. The late submission is accepted just before the next class with 2 points reductions, because the solution is demonstrated in the next class. |
|  |