Re-Examination in “Computer Graphics Programming”

5th Semester Medialogy, Aalborg University, Fall 2019

Instructions

- You have 3 hours to complete this examination.
- Any material and devices are allowed at the examination but all communication functions of all units have to be deactivated. SICT's rules for use of electronic devices in written exams apply.
- Write your name and student number on all sheets on which you write your answers. Write the number of the question before each answer (e.g. “Question 1a:”).
- There are 6 sections. Each section consists of some code and questions related to the code. Each question is worth 5 marks. In total, there are 20 questions.
- The maximum score is 100 marks. You must get at least 50 marks in order to pass.

Section 1

Consider the following code:

```
(1)  Shader "myshader" {
(2)      SubShader {
(3)          Pass {
(4)              CGPROGRAM
(5)              #pragma vertex vprog
(6)              #pragma fragment fprog
(7)
(8)              struct v2f {
(9)                  float4 tex : TEXCOORD0;
(10)                 float4 pos : SV_POSITION;
(11)             };
```

(code continues on next page)

```
(12)            float4 vprog(float4 p : POSITION, float4 t : TEXCOORD0) {
(13)               v2f o;
(14)               o.tex = t;
(15)               o.pos = UnityObjectToClipPos(p);
(16)               return o;
(17)            }
(18)            float4 fprog(v2f i) : TEXCOORD0 {
(19)               return i.t;
(20)            }
(21)            ENDCG
(22)         }
(23)      }
(24)   }
```

Question 1a

Specify the type (floating-point number, 3D vector, 4D vector, 4x4 matrix, or a struct) of each of the five variables "`p`", "`t`", "`o`", "`o.tex`", and "`i`".

Question 1b

There are three errors in the code. In which lines? What would be the correct lines?

Question 1c

Change the code to use the NORMAL vertex attribute instead of TEXCOORD0.

Question 1d

Change the code such that the x, y, and z object coordinates of all vertices are scaled by the factor 2.0.

Section 2

Consider the following code:

```
(1)  Shader "myshader" {
(2)     SubShader {
(3)        Pass {
(4)           CGPROGRAM
(5)           #pragma vertex myvert
(6)           #pragma fragment myfrag
(7)           uniform float4 a;
(8)           uniform float4 b;
(9)           float4 myvert(float4 p : POSITION) : SV_POSITION {
(10)             float4x4 m = UNITY_MATRIX_MV;
(11)             float4 q = mul(m, p + a);
(12)             float4 result = mul(UNITY_MATRIX_P, q + b);
(13)             return result;
(14)          }
(15)          float4 myfrag() : COLOR {
(16)             return float4(1.0, 1.0, 1.0, 1.0);
(17)          }
(18)          ENDCG
(19)       }
(20)    }
(21) }
```

Question 2a

From which and to which coordinate system does the matrix **`UNITY_MATRIX_P`** transform vectors? In which coordinate system are the three vectors in **`result, a,`** and **`b`** defined (or assumed to be defined)? (The options for all coordinate systems are: object coordinates, world coordinates, view coordinates, or clip coordinates)

Question 2b

Change the code to add **`a`** in view coordinates, i.e., the code should assume that **`a`** is specified in view coordinates and apply appropriate transformations for this case.

Question 2c

The components of m can be accessed as m[0][0], m[0][1], m[0][2], m[0][3], m[1][0], m[1][1], m[1][2], m[1][3], m[2][0], m[2][1], m[2][2], m[2][3], m[3][0], m[3][1], m[3][2], m[3][3]. The components of a can be accessed as a.x, a.y, a.z, and a.w. The components of p can be accessed as p.x, p.y, p.z, and p.w. In terms of these components, what is the y component of `mul(m, p + a)`?

Question 2d

Change line (12) to use the function `UnityObjectToClipPos()` to compute the same result.

Section 3

Consider the following code:

```
(1)  Shader "myshader" {
(2)     SubShader {
(3)        Pass {
(4)           Tags { "LightMode" = "ForwardBase" }
(5)           CGPROGRAM
(6)           #pragma vertex vert
(7)           #pragma fragment frag
(8)           float4 vert(float4 vertex : POSITION) : SV_POSITION {
(9)              return UnityObjectToClipPos(vertex);
(10)          }
(11)          float4 frag(void) : COLOR {
(12)              return float4(0.0, 0.0, 0.0, 1.0);
(13)          }
(14)          ENDCG
(15)       }
(16)       Pass {
(17)          Tags { "LightMode" = "ForwardAdd" }
(18)          Blend One One
(19)          CGPROGRAM
(20)          #pragma vertex vert
(21)          #pragma fragment frag
(22)          uniform float4 _LightColor0;
```

(code continues on next page)

```
(23)            struct vInput {
(24)               float4 vertex : POSITION;
(25)               float3 normal : NORMAL;
(26)            };
(27)            struct vOutput {
(28)               float4 pos : SV_POSITION;
(29)               float4 p : TEXTURE0;
(30)               float3 norm : TEXTURE1;
(31)            };
(32)            vOutput vert(vInput i) {
(33)               vOutput o;
(34)               o.pos = UnityObjectToClipPos(i.vertex);
(35)               o.p = mul(UNITY_MATRIX_MV, i.vertex);
(36)               o.norm = normalize(mul(UNITY_MATRIX_IT_MV,
(37)                  float4(i.normal, 0.0)).xyz);
(38)               return o;
(39)            }
(40)            float4 frag(vOutput o) : COLOR {
(41)               float3 light = mul(UNITY_MATRIX_V, _WorldSpaceLightPos0);
(42)               float3 view = normalize(mul(UNITY_MATRIX_V,
(43)                  _WorldSpaceCameraPos) - o.p);
(44)               float3 refl = reflect(-normalize(light), normalize(o.norm));
(45)               float3 col = pow(max(0.0, dot(view, refl)), 10.0) *
(46)                   _LightColor0.rgb * float3(0.0, 0.0, 1.0);
(47)               return float4(col, 1.0);
(48)            }
(49)            ENDCG
(50)         }
(51)      }
(52)   }
```

Question 3a

Which component or components of the Phong reflectance model (also known as Phong reflection model) is calculated in the second pass: specular, diffuse, ambient or none? For what kind of light source: ambient light, directional light or point light? What kind of attenuation is implemented: none, linear, quadratic, or cubic? In which coordinate system is the lighting computed (object coordinates, world coordinates, view coordinates, or clip coordinates)? Is the lighting computed in the vertex or fragment shader?

Question 3b

Change the code such that the light source is a point light source with linear attenuation.

Question 3c

Change the code to compute the same lighting in object coordinates.

Question 3d

Change the code to compute the diffuse reflection of the Phong reflectance model.

Section 4

Consider the following code:

```
(1)  Shader "myshader" {
(2)     SubShader {
(3)        Pass {
(4)           Tags { "LightMode" = "ForwardBase" }
(5)           CGPROGRAM
(6)           #pragma vertex vert
(7)           #pragma fragment frag
(8)           float4 vert(float4 vertex : POSITION) : SV_POSITION {
(9)              return UnityObjectToClipPos(vertex);
(10)          }
(11)          float4 frag(void) : COLOR {
(12)              return float4(0.0, 0.0, 0.0, 1.0);
(13)          }
(14)          ENDCG
(15)       }
(16)       Pass {
(17)          Tags { "LightMode" = "ForwardAdd" }
(18)          Blend One One
(19)          CGPROGRAM
(20)          #pragma vertex vert
(21)          #pragma fragment frag
(22)          uniform float4 _LightColor0;
(23)          struct vInput {
(24)             float4 vertex : POSITION;
(25)             float3 normal : NORMAL;
(26)          };
(27)          struct vOutput {
(28)             float4 pos : SV_POSITION;
(29)             float4 pos2 : TEXTURE0;
(30)             float3 norm : TEXTURE1;
(31)          };
```

(code continues on next page)

```
(32)          vOutput vert(vInput i) {
(33)              vOutput o;
(34)              o.pos = UnityObjectToClipPos(i.vertex);
(35)              o.pos2 = mul(unity_ObjectToWorld, i.vertex);
(36)              o.norm = normalize(mul(float4(i.normal, 0.0),
(37)                 unity_WorldToObject).xyz);
(38)              return o;
(39)           }
(40)           float4 frag(vOutput o) : COLOR {
(41)              float3 light = _WorldSpaceLightPos0.xyz - o.pos2.xyz;
(42)              float3 col = _LightColor0.rgb/length(light)/length(light) *
(43)                  max(0.0, dot(normalize(light), normalize(o.norm)));
(44)              return float4(col, 1.0);
(45)           }
(46)           ENDCG
(47)        }
(48)     }
(49)  }
```

Question 4a

Which component of the Phong reflectance model (also known as Phong reflection model) is calculated in the second pass: specular, diffuse, ambient or none? For what kind of light source: ambient light, directional light or point light? What kind of attenuation is implemented: none, linear, quadratic, or cubic? In which coordinate system is the lighting computed (object coordinates, world coordinates, view coordinates, or clip coordinates)? Is the lighting computed in the vertex or fragment shader?

Question 4b

In line 36, the vertex shader writes a normalized vector into “o.norm.” The fragment shader normalizes the value again with “normalize(o.norm)” in line 43. Is the second call to “normalize” superfluous? Briefly explain your answer.

Question 4c

Change the code to compute the lighting in the vertex shader.

Question 4d

Summarize the changes that are necessary for lighting of a two-sided surface. (You may provide the code or summarize the required changes in a short list without the full code.)

Section 5

Consider the following code:

```
(1)  Shader "myshader" {
(2)     Properties {
(3)        _Tex1 ("Texture", 2D) = "white" {}
(4)        _Tex2 ("Texture", 2D) = "white" {}
(5)     }
(6)     SubShader {
(7)        Pass {
(8)           Cull Front
(9)           CGPROGRAM
(10)          #pragma vertex vert
(11)          #pragma fragment frag
(12)          struct vOut {
(13)             float4 pos: SV_POSITION;
(14)             float4 texcoord: TEXCOORD0;
(15)          };
(16)          uniform sampler2D _Tex1;
(17)          uniform sampler2D _Tex2;
(18)          vOut vert(float4 p : POSITION, float4 t: TEXCOORD) {
(19)             vOut result;
(20)             result.pos = UnityObjectToClipPos(p);
(21)             result.texcoord = t;
(22)             return result;
(23)          }
(24)          float4 frag(vOut i) : COLOR {
(25)              float4 c2 = tex2D(_Tex2, i.texcoord.xy);
(26)              if (c2.a < 0.5) { discard; }
(27)              return c2;
(28)          }
(29)          ENDCG
(30)       }
(31)
```

(code continues on next page)

```
(32)        Pass {
(33)           Cull Back
(34)           CGPROGRAM
(35)           #pragma vertex vert
(36)           #pragma fragment frag
(37)           struct vOut {
(38)              float4 pos: SV_POSITION;
(39)              float4 texcoord: TEXCOORD0;
(40)           };
(41)           uniform sampler2D _Tex1;
(42)           uniform sampler2D _Tex2;
(43)           vOut vert(float4 p : POSITION, float4 t: TEXCOORD) {
(44)              vOut result;
(45)              result.pos = UnityObjectToClipPos(p);
(46)              result.texcoord = t;
(47)              return result;
(48)           }
(49)           float4 frag(vOut i) : COLOR {
(50)               float4 c1 = tex2D(_Tex1, i.texcoord.xy);
(51)               if (c1.a < 0.5) { discard; }
(52)               return c1;
(53)           }
(54)           ENDCG
(55)        }
(56)     }
(57)  }
```

Question 5a

This is a shader for a two-sided surface with different textures for the front-faces and back-faces. It uses the alpha component of each texture to discard fragments with alpha < 0.5. Since different textures are used for front-faces and back-faces, a point that is discarded on a front-face might not be discarded when it is on a back-face and vice versa, which doesn't make physical sense. Change the code such that the conditional discard uses the same alpha for front-faces and for back-faces.

Question 5b

Change the code to use blending instead of the discard instructions.

Section 6

Consider the following code:

```
(1)  Shader "CustomShader" {
(2)     Properties {
(3)        _Tex ("Texture", 2D) = "white" {}
(4)     }
(5)     SubShader {
(6)        Pass {
(7)           CGPROGRAM
(8)           #pragma vertex vert
(9)           #pragma fragment frag
(10)          uniform sampler2D _Tex;
(11)          struct appdata{
(12)             float4 position : POSITION;
(13)             float3 normal : NORMAL;
(14)             float4 texcoord : TEXCOORD0;
(15)          };
(16)          struct v2f {
(17)             float4 pos : SV_POSITION;
(18)             float4 color : COLOR;
(19)          };
(20)          v2f vert(appdata vIn) {
(21)             v2f vOut;
(22)             vOut.color = tex2Dlod(_Tex, vIn.texcoord);
(23)             float4 offset = float4(vIn.normal * sin(_Time.y), 0.0);
(24)             vOut.pos = UnityObjectToClipPos(vIn.position + offset);
(25)             return vOut;
(26)          }
(27)          float4 frag(v2f fIn) : COLOR {
(28)             return fIn.color;
(29)          }
(30)          ENDCG
(31)       }
(32)    }
(33) }
```

Question 6a

Describe the purpose served by the computations in lines 23 to 24. ("_Time.y" is a built-in shader variable that specifies the time since a level has been loaded.). Change the shader such that it displaces vertices only along the y axis in object coordinates.

Question 6b

The code computes an offset for each vertex (in the "offset" variable). Change the code to include a property and uniform variable that allows to scale the offset (by multiplying the offset with the value of the new uniform variable).