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)              uniform float s;
(9)              uniform float4x4 a;
(10)             float4 vpro(float4 p : POSITION) : SV_POSITION
(11)             {
(12)                 return UnityObjectToClipPos(a * p);
(13)             }
```

(code continues on next page)

```
(14)            float4 fprog() : COLOR
(15)            {
(16)               float3 color = float3(0.0, 1.0, 1.0);
(17)               float alpha = 1.0;
(18)               return float4(s * color; alpha);
(19)            }
(20)            ENDCG
(21)         }
(22)     }
(23)  }
```

Question 1a

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

Answer 1a

4D vector, float, 3D vector, 4x4 matrix, float (1 mark for each)

Question 1b

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

Answer 1b

line (5): **`#pragma vertex vpro`**
or line (10): **`#float4 vprog(`**… (1 mark)
line (12): **`return UnityObjectToClipPos(mul(a, p));`**
or line (9): **`uniform float a;`** (2 marks)
line (18): **`return float4(s * color, alpha);`** (2 marks)

Question 1c

Which color does "**`float3(0.0, 1.0, 1.0)`**" in line (16) represent (red, green, blue, yellow, magenta, cyan, or white)? If "`s`" is set to 0.5, what are the RGB components of "**`s * color`**"?

Answer 1c

cyan (2 marks); RGB: 0, 0.5, 0.5, (3 marks)

Question 1d

Change the code of the fragment shader to always return a vector that represents yellow.

Answer 1d

`line (18): return float4(1.0, 1.0, 0.0, 1.0);` (5 marks; one less for each mistake)

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)             float4 result = UnityObjectToClipPos(a + b * p);
(11)             return result;
(12)          }
(13)          float4 myfrag() : COLOR {
(14)             return float4(1.0, 1.0, 1.0, 1.0);
(15)          }
(16)          ENDCG
(17)       }
(18)    }
(19) }
```

Question 2a

From which and to which coordinate system is the function **`UnityObjectToClipPos()`** transforming vectors? In which coordinate systems are the three vectors in **`p, a,`** and **`result`** defined (or assumed to be defined)? (The options for all coordinate systems are: object coordinates, world coordinates, view coordinates, or clip coordinates)

Answer 2a

`object to clip (2 marks); p: object (1 mark), a: object (1 mark); result: clip (1 mark)`

Question 2b

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

Answer 2b

```
(10) float4 result = UnityObjectToClipPos(mul(unity_WorldToObject, a)
+ b * p);  (5 marks)
```

or:

```
(10) float4 result = mul(UNITY_MATRIX_P, mul(UNITY_MATRIX_V,
mul(unity_ObjectToWorld, b * p) + a));
(5 marks)
```

Question 2c

The components of p can be accessed as p.x, p.y, p.z, and p.w. The components of a can be accessed as a.x, a.y, a.z, and a.w. The components of b can be accessed as b.x, b.y, b.z, and b.w. In terms of these components, what is the z component of **`a + b * p`**?

Answer 2c

```
a.z + b.z * p.z (5 marks)
```

Question 2d

The appearance of a shaded surface does not change if line (14) is changed to "**`return float4(2.0, 1.0, 1.0, 1.0);`**" nor does it change if line (14) is changed to "**`return float4(1.0, 1.0, 1.0, 0.0);`**". Explain for both changes why they do not change the appearance.

Answer 2d

Values greater than 1.0 are clamped to 1.0. (2 marks)

```
Blending is not activated, thus, the alpha channel is not used. (3
marks)
```

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;
(23)          struct vInput {
(24)             float4 vertex : POSITION;
(25)             float3 normal : NORMAL;
(26)          };
(27)          struct vOutput {
(28)             float4 pos1 : 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.pos1 = UnityObjectToClipPos(i.vertex);
(35)               o.pos2 = i.vertex;
(36)               o.norm = i.normal;
(37)               return o;
(38)            }
(39)            float4 frag(vOutput o) : COLOR {
(40)               float3 light = mul(unity_WorldToObject, _WorldSpaceLightPos0)
(41)                  - o.pos2;
(42)               float3 view = normalize(mul(unity_WorldToObject,
(43)                  _WorldSpaceCameraPos) - o.pos2);
(44)               float3 refl = reflect(-normalize(light), normalize(o.norm));
(45)               float3 col = dot(normalize(o.norm), normalize(light)) *
(46)                   _LightColor0.rgb / length(light);
(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?

Answer 3a

Diffuse (1 mark); point (1 mark); linear (1 mark); object coordinates (1 mark); fragment shader (1 mark)

Question 3b

Change the code such that there is no attentuation and the light source is a directional light source.

Answer 3b

```
(40) float3 light = mul(unity_WorldToObject, _WorldSpaceLightPos0).xyz;
(41)
(46) _LightColor0.rgb;
```

(5 marks, one less for each mistake)

Question 3c

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

Answer 3c

```
(40) float3 light = _WorldSpaceLightPos0 - mul(unity_ObjectToWorld, o.pos2);
(41)
(45) float3 col = dot(normalize(mul(float4(o.norm, 0.0), unity_WorldToObject).xyz),
(46) normalize(light)) * _LightColor0.rgb / length(light);
```

(5 marks; one less for each mistake)

Question 3d

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

Answer 3d

```
(45) float3 col = pow(max(0.0, dot(normalize(view), normalize(refl))), 50.0) *
```

(5 marks; one less for each mistake; attenuation doesn't matter; material color doesn't matter)

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)             float3 col : COLOR;
(30)          };
(31)
```

(code continues on next page)

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

Question 4a

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 which material color? 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)?

Answer 4a

Diffuse (1 mark); yellow (1 mark); point (1 mark); none (1 mark); world coordinates (1 mark)

Question 4b

The code computes the lighting in the vertex shader. Name one advantage and one disadvantage of computing the lighting in the vertex shader.

Answer 4b

```
Advantage: better performance; disadvantage: lower quality. (3 marks for one
       correct advantage or disadvantage; 5 for both)
```

Question 4c

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

Answer 4c

```
            struct vOutput {
                 float4 pos : SV_POSITION;
                 float3 light : TEXCOORD0;
                 float3 norm : TEXCOORD1;
            };

            vOutput vert(vInput i) {
                  vOutput o;
                  o.light = normalize((_WorldSpaceLightPos0 -
                        mul(unity_ObjectToWorld, i.vertex)).xyz);
                  o.norm = normalize(mul(float4(i.normal, 0.0),
                        unity_WorldToObject).xyz);
                  o.pos = UnityObjectToClipPos(i.vertex);
                  return o;
            }
            float4 frag(vOutput i) : COLOR{
                  float3 col = max(0.0, dot(normalize(i.light),
                      normalize(i.norm))) * float3(1.0, 1.0, 0.0) *
                      _LightColor0.rgb;
                  return float4(col, 1.0);
            }
            ENDCG
         }
      }
}
(5 marks, one less for each mistake)
```

Question 4d

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

Answer 4d

- add "Queue" = "Transparent" to tags of both passes (2 marks)
- add “Blend One OneMinusSrcAlpha” to first pass (2 marks)
- add “ZWrite Off” to both passes (1 mark)

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)        Tags { "Queue" = "Transparent" }
(8)        Pass {
(9)           Zwrite Off
(10)          Blend SrcAlpha OneMinusSrcAlpha
(11)          CGPROGRAM
(12)          #pragma vertex vert
(13)          #pragma fragment frag
(14)          struct vOut {
(15)             float4 pos: SV_POSITION;
(16)             float4 texcoord: TEXCOORD0;
(17)          };
(18)          uniform sampler2D _Tex1;
(19)          uniform sampler2D _Tex2;
(20)          vOut vert(float4 p : POSITION, float4 t: TEXCOORD) {
(21)             vOut result;
(22)             result.pos = UnityObjectToClipPos(p);
(23)             result.texcoord = t;
(24)             return result;
(25)          }
(26)          float4 frag(vOut i) : COLOR {
(27)              float4 c1 = tex2D(_Tex1, i.texcoord.xy);
(28)              float4 c2 = tex2D(_Tex2, i.texcoord.xy);
(29)              return float4(c1.rgb + c2.rgb, min(c1.a, c2.a));
(30)          }
(31)          ENDCG
(32)       }
(33)    }
(34) }
```

Question 5a

This is a shader for a (semi-)transparent surface with back-face culling. Which type of blending (straight alpha blending, premultiplied alpha blending, additive blending, or multiplicative blending) is used? Change the code such that the shader outputs a color for front-faces and for back-faces.

Answer 5a

straight alpha blending (2 marks)

“Cull Off” after line (8) (3 marks)

or: 2nd pass with “Cull Front”

Question 5b

The code combines the color of "_Tex2" with the color of "_Tex1". How are the color components combined? How is the opacity (alpha component) computed? (Specify the mathematical operations in both cases.)

Answer 5b

Color components are added (2 marks); the minimum of the opacities is used (3 marks)

Section 6

Consider the following code:

```
(1)  Shader "CustomShader" {
(2)     Properties {
(3)        _WaveTex ("Texture", 2D) = "white" {}
(4)     }
(5)     SubShader {
(6)        Pass {
(7)           CGPROGRAM
(8)           #pragma vertex vert
(9)           #pragma fragment frag
(10)          uniform sampler2D _WaveTex;
(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(_WaveTex, vIn.texcoord +
(23)                float4(_Time.y, 0, 0, 0));
(24)             float4 offset = float4(vIn.normal * vOut.color.a, 0.0);
(25)             vOut.pos = UnityObjectToClipPos(vIn.position + offset);
(26)             return vOut;
(27)          }
(28)          float4 frag(v2f fIn) : COLOR {
(29)             return fIn.color;
(30)          }
(31)          ENDCG
(32)       }
(33)    }
(34) }
```

Question 6a

Describe the purpose served by the computations in lines 22 to 25. ("_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 by twice the distance.

Answer 6a

The code reads a texture from a time-varying position and uses the alpha component to displace vertices by their surface normal vector. (2 marks)

```
(25) vOut.pos = UnityObjectToClipPos(vIn.position + 2.0 * offset);
```

(3 marks)

Question 6b

The code computes colors only for vertices. For each fragment, it interpolates between those colors. Change the code such that the fragment shader samples the same texture as the vertex shader and returns the sampled color.

Answer 6b

```
(18) float4 tex : TEXCOORD0;

(22) vOut.tex = vIn.texcoord + float4(_Time.y, 0, 0, 0);

(23) float4 color = tex2Dlod(_WaveTex, vIn.texcoord + float4(_Time.y, 0, 0, 0));

(29) return tex2Dlod(_WaveTex, fIn.tex);
```

(5 marks)