HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Computer Graphics

T- 511 – TGRA

Final Exam

Teacher: Kári Halldórsson

Date: 20. November, 2017

Time: 14:00 - 17:00

Helping materials: non-programmable calculator

Name: LAUSNIR

Kt.: ______________________

Answers can be given in English and/or Icelandic.

1. **(10%) Transparency**
How could one render a transparent object in OpenGL?
 - Where in the OpenGL pipeline would this affect the calculations (and very briefly how)?
 - What would one specifically have to consider, as a programmer using OpenGL, when rendering a transparent object, in order for the effect to appear as intended?

How:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
            GL_ONE_MINUS_SRC_ALPHA);
```

~~the~~
then make sure to have alpha < 1.

Effects:

In rasterization, after running the fragment shader, the color in gl_FragColor is mixed with the color already in the frame buffer at the corresponding pixel:

```
frame_buffer[x,y] = gl_FragColor.alpha * gl_FragColor
                  + (1 - gl_FragColor.alpha) * frame_buffer[x,y]
```

and the result is put in the frame buffer instead of putting gl_FragColor directly.

Considerations:

You have to render everything behind the transparent object <u>before</u> rendering the — " — object itself.

2. (10%) Shaders and lighting

Describe the difference between per-vertex lighting and per-fragment/per-pixel lighting.
In each case bear the following questions in mind:
What are the advantages and drawbacks of the method?
What calculations happen where, and what values are set to the final result?
How is the data processed between different parts of the calculations?

In per-vertex lighting the final color for the vertex is calculated in the vertex shader. (lighting, lambert, phong, happens in vertex shader)
During rasterization the color is linearly interpolated over the pixels and the results put in the frame buffer.
This is faster than per-fragment lighting.

In per-fragment lighting relative directions are calculated as vectors in the vetex shader.
During rasterization the values of these vectors are ~~used to~~ linearly interpolated over the pixels and then used to calculate the final color for each pixel in the fragment shader. (lighting, lambert, phong, happens in fragment shader)
This looks much smoother and more detailed than per-vertex lighting.

3. (10%) Cohen-Sutherland Clipping

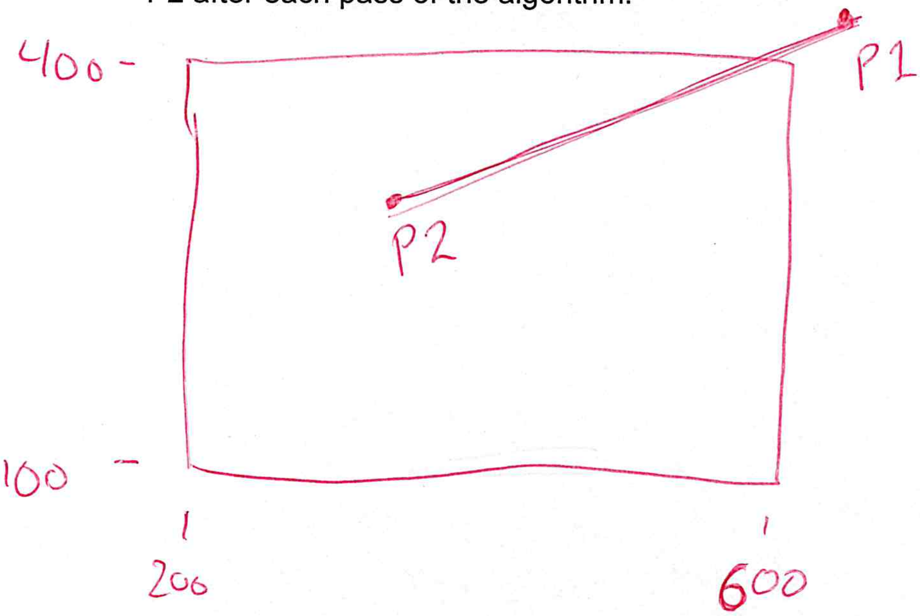
A clipping window has the following geometry:

Window(left, right, bottom, top) = (200, 600, 100, 400)

A line with the following end points is drawn in the world:

P1: (650, 450)
P2: (350, 300)

Show how the Cohen-Sutherland clipping algorithm will clip ~~these~~ this line~~s~~ and what ~~their~~ its final endpoint~~s~~, if any, ~~are~~ is. Show the coordinate values of P1 and P2 after each pass of the algorithm.



left, right, above, below

code:

l	r	a	b

Pass 1:

code 1: $\boxed{0\ 1\ 1\ 0}$

code 2: $\boxed{0\ 0\ 0\ 0}$

no trivial accept
no trivial reject

$delx = P1.x - P2.x = 650 - 350 = \underline{300}$

$dely = P1.y - P2.y = 450 - 300 = \underline{150}$

i) P1 is outside

if above:

$P1.x = P1.x + (\cancel{top} - P1.y) \cdot \frac{delx}{dely} = 650 + (400-450) \cdot \frac{300}{150}$

$= 650 - 50 \cdot 2$

$= \underline{550}$

$P1.y = top = \underline{400}$

$\underline{P1 = (550, 400)}$ $P2 = (350, 300)$

Pass 2:

code1: $\boxed{0\,0\,0\,0}$

code2: $\boxed{0\,0\,0\,0}$

Trivial accept!!

4. (15%) Shader programming

On the following two pages there are shaders for use in an OpenGL pipeline, a *vertex shader* and a *fragment shader*. You can add both *uniform* and *varying* variables to these shaders above their main() function definition and you can add any number of variables and code inside the main functions.

The following types can be used:

float:	a floating point number
vec4:	a 4 coordinate vector, used for positions, directions and colors
mat4:	a 4x4 matrix

The following GLSL functions can be used:

float dot(vec4 v1, vec4 v2):	Returns the dot product of two vectors
vec4 normalize (vec4 v):	Returns the normalized vector
float length(vec4 v):	Returns the length of the vector
vec4 cross(vec4 v1, vec4 v2):	Returns the cross product of two vectors
float max(float a, float b):	Returns the higher value
float min(float a, float b):	Returns the lower value
float pow(float num, float exp):	Returns *num* to the power of *exp*

+, -, *, / are done component-wise on *vec4* but as matrix multiplications when one or both sides are *mat4*.

Variables are already defined for the attributes *a_position* and *a_normal* for each vertex that will be sent through the pipeline.

The current shaders both have the same initial definitions at this point. **Strike out those that are unnecessary** in each shader and add whatever definitions you need for each of the following problems.

a) **(5%)** Add variables and calculations needed to transform the position attribute (vertices) to global coordinates, eye coordinates and clip coordinates and set the value for the built-in output variable *gl_Position*.

b) **(10%)** Add variables and calculations needed for the shaders to do per-fragment ***diffuse*** lighting. Diffuse lighting (not specular, ambient or emissive) for a ***single** light source* is enough. Set the final color value to the built-in output variable *gl_FragColor*.

Vertex shader:

```
attribute vec3 a_position;
attribute vec3 a_normal;

a) uniform mat4 u_modelMatrix;
   uniform mat4 u_viewMatrix;
   uniform mat4 u_projectionMatrix;

uniform vec4 u_lightPosition;
```

~~uniform vec4 u_lightColor;~~

~~uniform vec4 u_materialAmbient;~~
~~uniform vec4 u_materialDiffuse;~~
~~uniform vec4 u_materialSpecular;~~
~~uniform float u_materialShininess;~~

```
b) varying vec4 v_normal;
   varying vec4 v_s;

void main()
{
a)     vec4 position = vec4(a_position, 1);
       vec4 normal = vec4(a_normal, 0);
       position = u_modelMatrix * position;

b)     v_normal = u_modelMatrix * normal;

       v_s = u_lightPosition - position;

a)     position = u_viewMatrix * position;
       glPosition = u_projectionMatrix * position;
}
```

Fragment shader:

~~attribute vec3 a_position;~~
~~attribute vec3 a_normal;~~

~~uniform vec4 u_lightPosition;~~
uniform vec4 u_lightColor;

~~uniform vec4 u_materialAmbient;~~
uniform vec4 u_materialDiffuse;
~~uniform vec4 u_materialSpecular;~~
~~uniform float u_materialShininess;~~

b) varying vec4 v_normal;
varying vec4 v_s;

void main()
{

b) float lambert = max (0,
dot (v_nomal , v_s) /(length (v_normal) * length(s))
;

gl_FragColor = lambert * u_lightColor * u_materialDiffuse;

}

5. (25%) Matrices and transformations

a) (10%)
A camera is set up to be positioned in (5, 3, 7)
looking at the point (0, 0, 0).
It has an up vector (0,1,0).
Set up the values in a matrix that represents this position and orientation of a camera.
Which matrix in your shader should be set to these values?

$eye = (5, 3, 7) \quad center = (0, 0, 0) \quad up = (0,1,0)$

$n = eye - center = (5,3,7)$

$u = up \times n = (0,1,0) \times (5,3,7)$
$= (7, 0, -5)$

$v = n \times u = (5,3,7) \times (7,0,-5)$
$= (-15, 49+25, -21) = (-15, 74, -21)$

REMEMBER TO NORMALIZE!!!

$|n| = \sqrt{25+9+49} = \sqrt{83} \qquad \hat{n} = \left(\frac{5}{\sqrt{83}}, \frac{3}{\sqrt{83}}, \frac{7}{\sqrt{83}}\right)$

$|u| = \sqrt{49+25} = \sqrt{74} \qquad \hat{u} = \left(\frac{7}{\sqrt{74}}, 0, \frac{-5}{\sqrt{74}}\right)$

$|v| = \sqrt{225+5476+441} = \sqrt{6142} \qquad \hat{v} = \left(\frac{-15}{\sqrt{6142}}, \frac{74}{\sqrt{6142}}, \frac{-21}{\sqrt{6142}}\right)$

$\boxed{-eye \circ u = 0} \quad \boxed{-eye \circ v = \frac{-5\cdot(-15)+(-3)\cdot 74+(-7)\cdot(-21)}{\sqrt{6142}} = 0}$

$-eye \circ n = \frac{-5\cdot 5+(-3)\cdot 3+(-7)\cdot 7}{\sqrt{83}} = \frac{-83}{\sqrt{83}} = -\sqrt{83}$

$$\text{View Matrix} = \begin{bmatrix} 7/\sqrt{74} & 0 & -5/\sqrt{74} & 0 \\ -15/\sqrt{6142} & 74/\sqrt{6142} & -21/\sqrt{6142} & 0 \\ 5/\sqrt{83} & 3/\sqrt{83} & 7/\sqrt{83} & -\sqrt{83} \\ 0 & 0 & 0 & 1 \end{bmatrix} \simeq \begin{bmatrix} 0{,}81 & 0 & -0{,}58 & 0 \\ -0{,}19 & 0{,}94 & -0{,}27 & 0 \\ 0{,}55 & 0{,}33 & 0{,}77 & -9{,}1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b) (5%)
The camera should have a field of view of 90°, an aspect ratio of 16:9, a near plane at 10 and a far plane at 110. Find the exact values for a matrix that calculates this camera.
Which matrix in your shader should be set to these values?

$fovy = 90 \quad ratio = 16/9 = 1{,}778 \quad near = 10, \; far = 110$

$top = N \cdot \tan\left(\frac{fovy}{2}\right) = 10 \cdot \tan(45°) = 10$

$bottom = -top = -10$

$right = top \cdot ratio = 10 \cdot 1{,}778 = 17{,}78$

$left = -right = -17{,}78$

$$\underline{\underline{Projection}} = \begin{bmatrix} \frac{2\cdot 10}{17{,}78-(-17{,}78)} & 0 & 0 & 0 \\ 0 & \frac{2\cdot 10}{10-(-10)} & 0 & 0 \\ 0 & 0 & \frac{-120}{100} & \frac{-2\cdot 10\cdot 110}{100} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0{,}56 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1{,}2 & 22 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

c) (5%)
Vertex data should be drawn into a coordinate frame that has been translated by (1, 7, 3) and then rotated by 150° about the y-axis. Represent this coordinate frame in a matrix. Which matrix would this commonly be?

$$\underline{\underline{\text{Model matrix}}} = \begin{bmatrix} -0{,}866 & 0 & 0{,}5 & 1 \\ 0 & 1 & 0 & 7 \\ -0{,}5 & 0 & -0{,}866 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

d) (5%)
A vertex is run through the vertex shader.
It has the position values (2, 3, 1).
Given the matrix values calculated in parts a, b & c, what values will the vertex shader set to gl_Position?
Will this vertex be within the viewing volume and thus (other tests notwithstanding) be rendered as part of the final image? Explain.

$$\begin{bmatrix} -0{,}866 & 0 & 0{,}5 & 1 \\ 0 & 1 & 0 & 7 \\ -0{,}5 & 0 & -0{,}866 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -0{,}232 \\ 10 \\ -1{,}866 \\ 1 \end{bmatrix} \quad \text{(global)}$$

$$\begin{bmatrix} 0{,}81 & 0 & -0{,}58 & 0 \\ -0{,}19 & 0{,}94 & -0{,}27 & 0 \\ 0{,}55 & 0{,}33 & 0{,}77 & -9{,}1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -0{,}232 \\ 10 \\ -1{,}866 \\ 1 \end{bmatrix} = \begin{bmatrix} 0{,}89 \\ 9{,}95 \\ -7{,}36 \\ 1 \end{bmatrix} \quad \text{(eye)}$$

$$\begin{bmatrix} 0{,}56 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1{,}2 & 22 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0{,}89 \\ 9{,}95 \\ -7{,}36 \\ 1 \end{bmatrix} = \begin{bmatrix} 0{,}498 \\ 9{,}95 \\ 30{,}83 \\ 7{,}36 \end{bmatrix} = \begin{bmatrix} 0{,}498/7{,}36 \\ 9{,}95/7{,}36 \\ 30{,}83/7{,}36 \\ 1 \end{bmatrix} = \begin{bmatrix} 0{,}065 \\ 1{,}35 \\ 4{,}1 \\ 1 \end{bmatrix}$$

Will NOT be seen, both y and z (pseudodepth) > 1 will be clipped!

6. (10%) Vector intersections and reflections

A line has end points (9, 0) and (7, 4).
A particle starts at (3, 1) and travels along in the direction (6, 2).

a) (7%) In which point does the path of the particle cross the line?

$A = (3,1) \qquad c = (6,2)$

$B = (9,0) \qquad n = ((9,0)-(7,4))^{\perp} = (2;-4)^{\perp} = (4,2)$

$$t_{hit} = \frac{n \circ (B-A)}{n \circ c} = \frac{4 \cdot 6 + 2 \cdot (-1)}{4 \cdot 6 + 2 \cdot 2} = \frac{22}{28} = 0{,}786$$

$P_{hit} = A + t_{hit} \cdot c = (3,1) + 0{,}786 \cdot (6,2) = \underline{(7{,}716 ; 2{,}572)}$

$$(7{,}716 ; 2{,}572)$$

b) (3%) If the particle is made to bounce off the line, what will its new direction vector be?

$$r = c - 2 \cdot \frac{(c \circ n)}{n \circ n} \cdot n = (6,2) - 2 \cdot \frac{6 \cdot 4 + 2 \cdot 2}{4 \cdot 4 + 2 \cdot 2}(4,2)$$

$$= (6,2) - 2 \cdot \frac{28}{20}(4,2)$$

$$= (6,2) - (2{,}8 \cdot 4 ; 2{,}8 \cdot 2)$$

$$= (6 - 11{,}2 ; 2 - 5{,}6) = \underline{(-5{,}2 ; -3{,}6)}$$

7. **(10%) Bezier motion**

Scalars in bezier curves can be found by factoring Bernstein polynomials: $BL = ((1-t) + t)^L$ for a bezier curve with $L + 1$ control points.

The camera is moved along a bezier curve with 4 control points.
P1 = (7, 3, 2), **P2** = (18, 3, 5), **P3** = (15, 3, 8), **P4** = (9, 3, 11)

The motion should start 14 seconds after the program starts and it should end 24 seconds later, 38 seconds after the program starts.
What is the camera's position 22 seconds after the program started?

$$t = \frac{\text{currentTime} - \text{startTime}}{\text{endTime} - \text{startTime}} = \frac{22-14}{38-14} = \frac{8}{24} = \frac{1}{3}$$

$$P = (1-t)^3 \cdot P1 + 3 \cdot (1-t)^2 \cdot t \cdot P2 + 3 \cdot (1-t) t^2 \cdot P3 + t^3 \cdot P4$$

$$= \frac{8}{27} \cdot (7,3,2) + \frac{12}{27} \cdot (18,3,5) + \frac{6}{27} \cdot (15,3,8) + \frac{1}{27} \cdot (9,3,11)$$

$$= \left(\frac{8 \cdot 7 + 12 \cdot 18 + 6 \cdot 15 + 9}{27}, \frac{8 \cdot 3 + 12 \cdot 3 + 6 \cdot 3 + 3}{27}, \frac{8 \cdot 2 + 12 \cdot 5 + 6 \cdot 8 + 11}{27} \right)$$

$$= \left(\frac{371}{27}, \frac{81}{27}, \frac{135}{27} \right)$$

$$= \underline{\underline{(13{,}74, 3, 5)}}$$

8. (10%) Rasterization

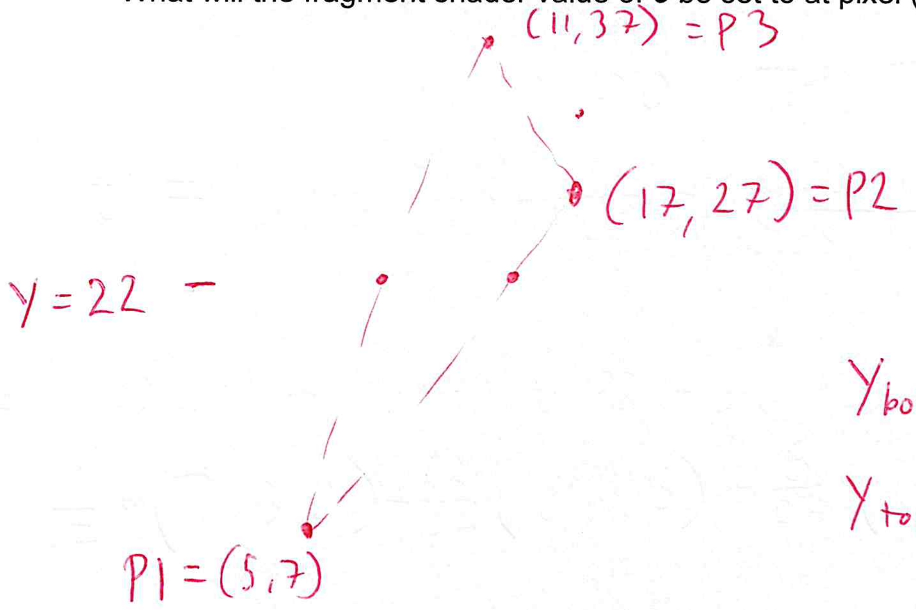
Three vertices of a triangle have been sent through the OpenGL pipeline. They each have an (x, y) pixel position as well as a varying color value ***c***.

P1: position = (5, 7), ***c*** = 0.4

P2: position = (17, 27), ***c*** = 0.0

P3: position = (11, 37), ***c*** = 1.0

What will the fragment shader value of ***c*** be set to at pixel (12, 22)?



$Y_{bott} = 7$

$Y_{top} = 37$

8

outer loop: $y = 22$

$X_{left} = lerp\left(5, 11, \frac{22-7}{37-7}\right) = 0{,}5 \cdot 5 + 0{,}5 \cdot 11 = 8$

$X_{right} = lerp\left(5, 17, \frac{22-7}{27-7}\right) = 0{,}25 \cdot 5 + 0{,}75 \cdot 17 = 14$

$C_{left} = lerp(0{,}4, 1{,}0, 0{,}5) = 0{,}5 \cdot 0{,}4 + 0{,}5 \cdot 1 = 0{,}7$

$C_{right} = lerp(0{,}4, 0{,}0, 0{,}75) = 0{,}25 \cdot 0{,}4 + 0{,}75 \cdot 0 = 0{,}1$

inner loop: $x = 12$

$frag\_value\_c[12, 22] = lerp\left(C_{left}, C_{right}, \frac{x - X_{left}}{X_{right} - X_{left}}\right)$

$= lerp\left(0{,}7 ; 0{,}1 ; \frac{12-8}{14-8}\right) = lerp\left(0{,}7; 0{,}1; \frac{2}{3}\right)$

$= \frac{1}{3} \cdot 0{,}7 + \frac{2}{3} \cdot 0{,}1 = \underline{0{,}3}$