COMP3011 Computer Graphics

Assessment 2

For assessment 2 your task is to program a 2D rasteriser. The program will be quite short and will be in one file.

You will need to write approximately 70 lines of C code, which will be split into functions.

You will need to implement

1. ClearColourBuffer(int r, int g, int b)
2. AssembleTriangles(float* verts, int n_verts, int* n_tris)
3. TransformToViewport(int width, int height, triangle* tri)
4. ComputeBarycentricCoordinates(float x, float y, triangle t, float& alpha, float& beta, float& gamma)
5. ShadeFragment(float alpha, float beta, float gamma, triangle t, int& r, int& g, int& b)

You will need to consider the following parts of the rendering pipeline

1. Vertex Specification (see chapter 2, pages 7 & 8)
2. Primitive Assembly (see chapter 2, pages 9 & 10)
3. NDC space (see chapter 6, page 17)
4. Viewport transformation (see chapter 6, page 19)
5. Screen space (see chapter 6, page 20)
6. Rasterisation (see chapter 2, pages 11-13)
7. Fragment Shader (see chapter 2, page 14)

You will need to submit to Moodle, a copy of the file containing the source code of your implementation. You only need to submit “COMP3011_ASSESSMENT_2.cpp”. Only submit one single file. When you test your program in the Visual Studio solution before submitting, make sure that it runs on the A32 PCs (or the GPU Virtual Desktop if you have my permission).

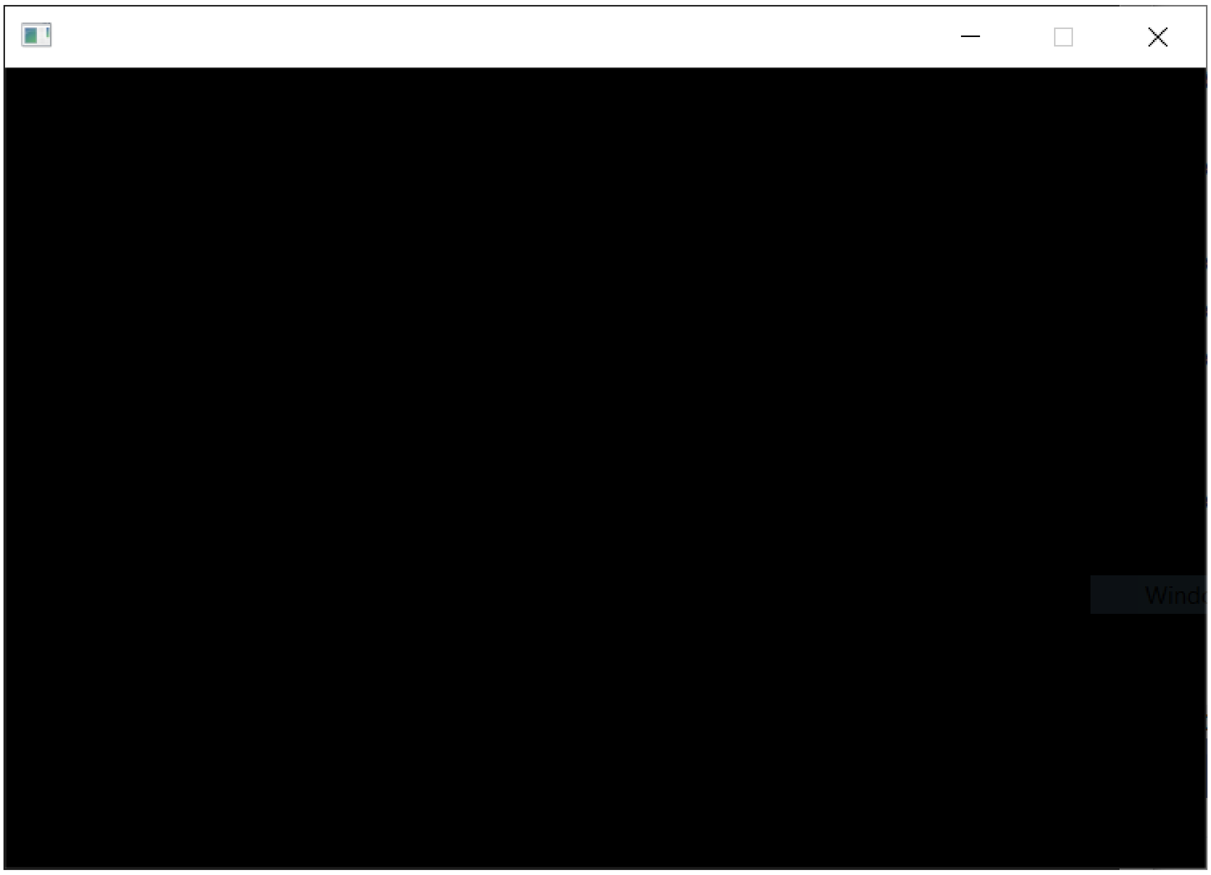
Setting up

Download the COMP3011_ASSESSMENT_2 zipped directory from Moodle.

Extract the files.

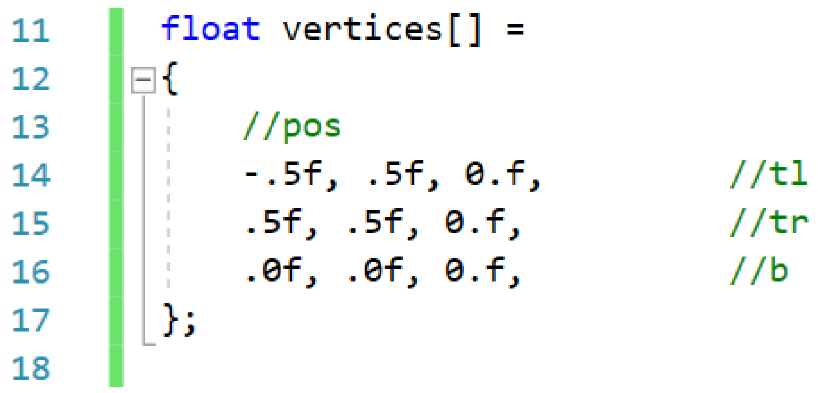
The solution does not need any of the GLFW or GLAD files from the lab exercises, it is self-contained, and just uses SDL. So you can put the directory anywhere.

Open up “COMP3011_ASSESSMENT_2.sln” and click on the Run Button and you should see an empty window



Get to know the files

At the top of COMP3011_ASSESSMENT_2.cpp you will see the vertex array



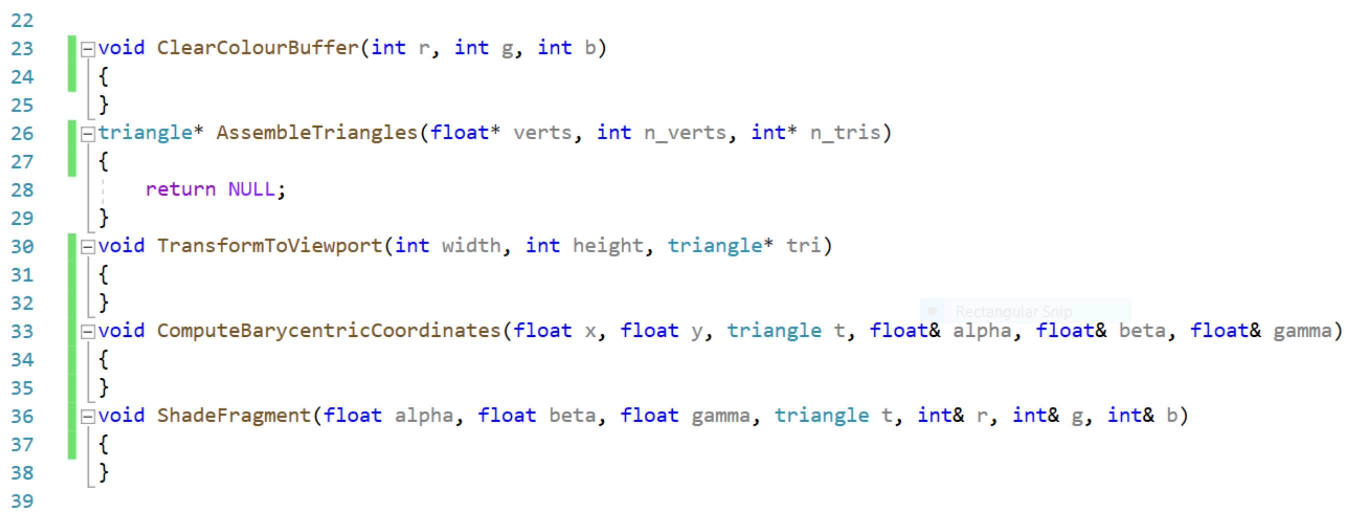
```
float vertices[] =
{
    //pos
    -.5f, .5f, 0.f,         //tl
    .5f, .5f, 0.f,          //tr
    .0f, .0f, 0.f,          //b
};

```

The vertex attributes are the vertex positions in NDC space, (-1 to 1 in X, Y and Z directions).

Since the rasteriser is for 2D, the z component should be 0 for all vertices.

Underneath you will see the empty functions that you will need to implement

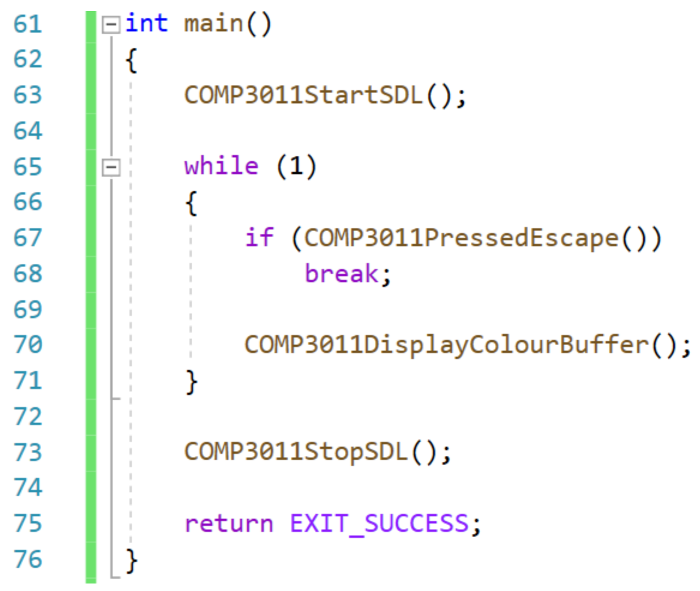


```

void ClearColourBuffer(int r, int g, int b)
{
}
triangle* AssembleTriangles(float* verts, int n_verts, int* n_tris)
{
    return NULL;
}
void TransformToViewport(int width, int height, triangle* tri)
{
}
void ComputeBarycentricCoordinates(float x, float y, triangle t, float& alpha, float& beta, float& gamma)
{
}
void ShadeFragment(float alpha, float beta, float gamma, triangle t, int& r, int& g, int& b)
{
}

```

At the bottom you will find the main() function



```
int main()
{
    COMP3011StartSDL();

    while (1)
    {
        if (COMP3011PressedEscape())
            break;

        COMP3011DisplayColourBuffer();
    }

    COMP3011StopSDL();

    return EXIT_SUCCESS;
}
```

1. COMP3011StartSDL() initialises SDL and creates a window.
2. COMP3011PressedEscape() returns true if the user pressed the escape key
3. COMP3011DisplayColourBuffer() will copy whatever is in the colour buffer to the SDL window.
4. COMP3011StopSDL() closes SDL

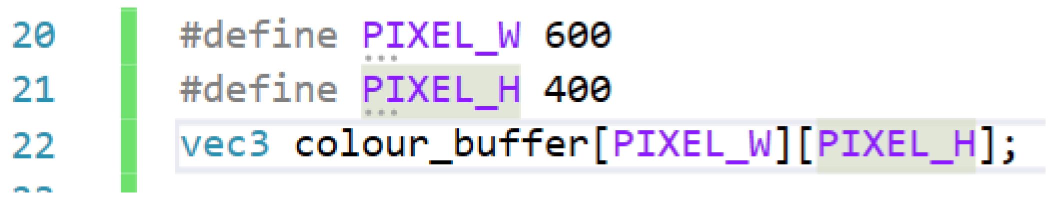
You do not need to change any function implementation or function call of any function starting with COMP3011***

Completing the assessment

It should be obvious the order the functions need to be called to rasterise a triangle.

The Colour Buffer

The best thing to do first is get your head around how the colour buffer works.



```
#define PIXEL_W 600
#define PIXEL_H 400
vec3 colour_buffer[PIXEL_W][PIXEL_H];
```

It is a 2D array of vec3 elements. It is as large as the display `PIXEL_W` x `PIXEL_H`.

The vec3 struct is defined in "DoNotModify.h", which as the name suggests, you do not need to and should not modify.

vec3 has an x, y, and z component. It is used by the vertex position XYZ components and also by the colour RGB components. For the colour buffer the x corresponds to the R, y to G, and z to B values which will be displayed at a given pixel.

In the main() function, you will find `COMP3011DisplayColourBuffer()`. It is defined in "DoNotModify.h", It copies each pixel colour from the colour buffer to the SDL display. Note that SDL uses [0..255] for the RGB components of a colour.

To get you started, in the main() function, try setting the colour of a given pixel in the colour_buffer and verify that it is copied to the display when you run the program.

Implement the function

```
void ClearColourBuffer(int r, int g, int b)
```

which clears the whole colour buffer to a colour specified by the `r, g, b` arguments

This should be about 10 lines of code.

Triangles

Next implement AssembleTriangles()

```
triangle* AssembleTriangles(float* verts, int n_verts, int* n_tris)
```

`verts` is the array of vertex attributes, `n_verts` is the number of vertices, `n_tris` will hold the number of triangles.

This function should allocate memory for `triangles`, copy the vertex attributes to each triangle vertex position, update the value of `n_tris` and return the `triangles`.

This should be about 20 lines of code.

Viewport transformation

Next implement TransformPixelToNDC ()

```
void TransformToViewport(int width, int height, triangle* tri)
```

This function should transform the vertex positions from NDC to pixel coordinates.

This should be about 10 lines of code.

Barycentric Coordinates

We need to calculate the barycentric coordinates and use them to test if a pixel is inside a triangle.

Implement the function ComputeBarycentricCoordinates()

```
void ComputeBarycentricCoordinates(float x, float y, triangle t, float& alpha, float& beta, float& gamma)
```

`x` and `y` are the current pixel coordinate, `t` is the triangle to be tested, `alpha beta` and `gamma` will store the barycentric coordinates.

This should be about 10 lines of code.

You already have practiced this calculation twice, once in Lab2 and once in Assessment 1.

Fragment Shader

For this coursework the fragment shader should simply return a colour

```
void ShadeFragment(float alpha, float beta, float gamma, triangle t, int& r, int& g, int& b)
```

Just ignore the barycentric coordinates `alpha, beta, gamma,` and triangle `t` .

This should be 1 line of code.

Rasterisation

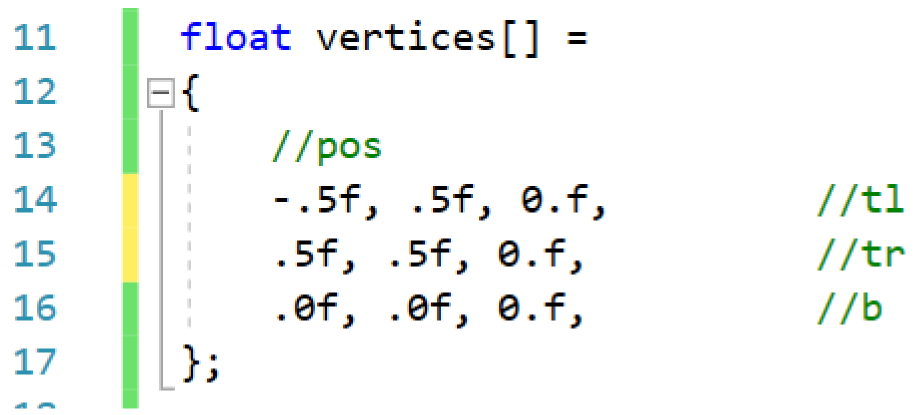
You need to implement the rasteriser. This is the easy part once you have everything else set up.

This should be about 20 lines of code.

SDL screen space

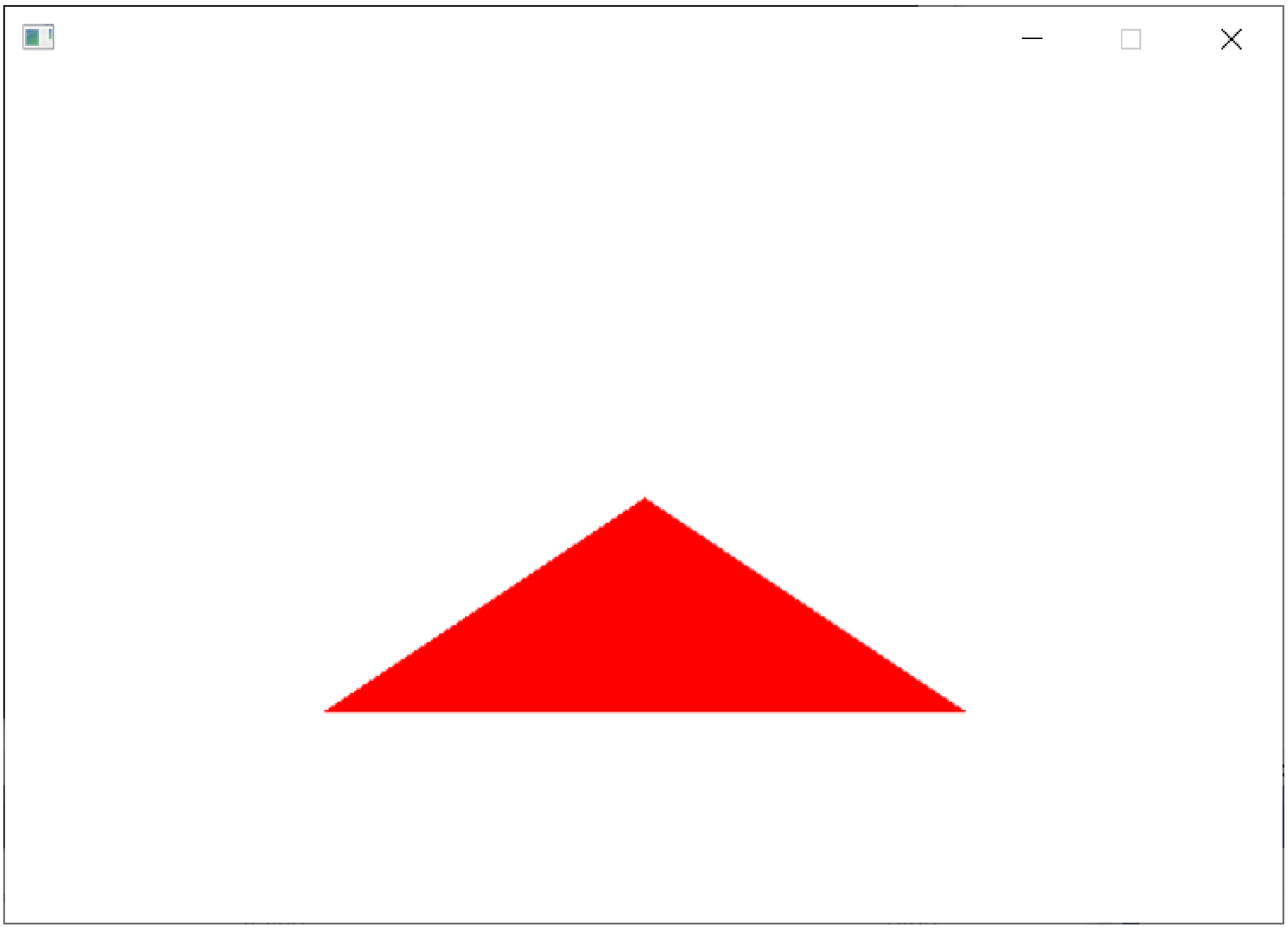
When you first get your rasteriser working, you will probably notice that the triangles are drawn upside down.

If you have vertices specified as

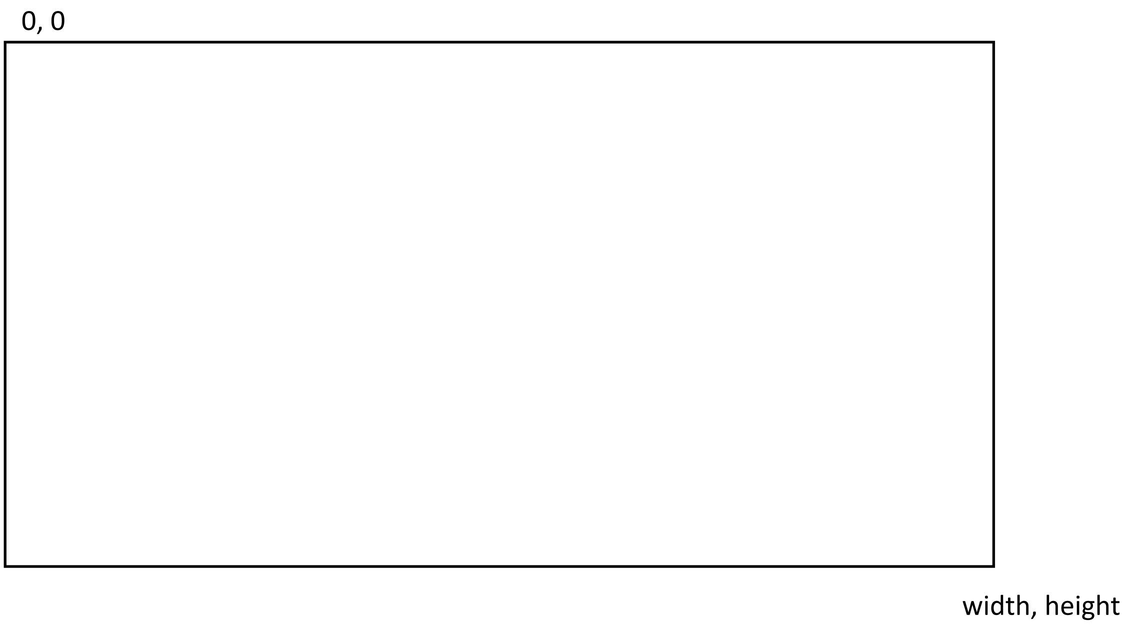


```
float vertices[] =
{
    //pos
    -.5f, .5f, 0.f,         //tl
    .5f, .5f, 0.f,          //tr
    .0f, .0f, 0.f,          //b
};
```

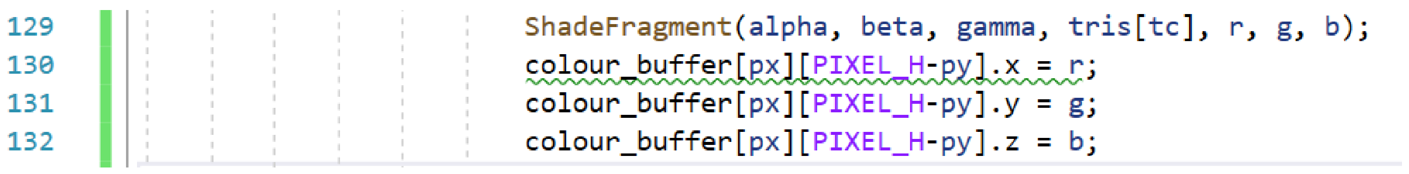
The triangle might be rendered like this



This is because in SDL the screen space is



The correct way to fix this is after you have called your fragment shader



```
                        ShadeFragment(alpha, beta, gamma, tris[tc], r, g, b);
                        colour_buffer[px][PIXEL_H-py].x = r;
                        colour_buffer[px][PIXEL_H-py].y = g;
                        colour_buffer[px][PIXEL_H-py].z = b;
```

Marking Scheme

The following marking scheme will be used

Test #	**Test**	**Marks**	
1	Is at least 1 pixel coloured in a predictable manner?	1	Running (2)
2	AND does the program run without crashing?	1	
3	Is `ClearColourBuffer()` implemented correctly?	1	Clear Screen (2)
4	Can the display be cleared to a specific colour?	1	
5	Is `AssembleTriangles()` implemented correctly?	1	Triangle Rasterised (9)
6	Is `TransformToViewport()` implemented correctly?	1	
7	Is `ComputeBarycentricCoordinates()` implemented correctly?	3	
8	Is rasterisation implemented correctly?	1	
9	Is a triangle rendered at the correct position	1	
10	Is the triangle the right way up?	1	
11	Can the triangle colour be specified in `ShadeFragment()`	1	Shader (1)
12	Can an additional triangle be rendered	2	Bonus (2)

Marking and Demos

You are expected to demo your rasteriser to a teacher who will ask you questions to verify that you are awarded the correct mark. The demos will happen in A32 (or the remote channel in Teams) shortly after the submission deadline.