Task 2: The Affine Transformation Matrix

b) V M

X
Y
Z
1

A	B	0	C
D	E	0	F
0	0	1	0
0	0	0	1

For A:

i) Flips the image over Y-axis, looks like rotating
ii) Scaling
iii) X-axis
iv) Manipulates X as Ma*V= [Ax, y, z, 1],

A	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

For B:

i) Tilts the image sideways
ii) Shearing
iii) X-axis
iv) Manipulates X with respect to the y-axis: M*V = [x+By, y, z,1]

1	B	0	0
0	1	0	0
0	0	1	0
0	0	0	1

For C:

i) Slides the figures along the x-axis
ii) Translation
iii) X-axis
iv) Manipulates X as M*V = [x+C, y, z, 1]

1	0	0	C
0	1	0	0
0	0	1	0
0	0	0	1

For D:

i) Tilts the image up and down, giving off a wavy feel
ii) Shearing
iii) Y-axis
iv) Manipulates X with respect to the y-axis: M*V = [x, y+Dx, z,1]

1	0	0	0
D	1	0	0
0	0	1	0
0	0	0	1

For E:

i) Flips the image over the X-axis, looks like rotating
ii) Scaling
iii) Y-axis
iv) Manipulates Y as Ma*V = [x, Ey, z,1]
v)

1	0	0	0
0	E	0	0
0	0	1	0
0	0	0	1

For F:

v) Slides the figures along the Y-axis
vi) Translation
vii) Y-axis
viii) Manipulates Y as M*V = [x+C, y, z,1]

1	0	0	0
0	1	0	F
0	0	1	0
0	0	0	1

Task 3d: Combinations of Transformations

i) This can be done. Scaling up the square, flipping it and then shearing it along y-axis causes this. By flipping I mean rotating around its own axis, which can be done by moving (translating) to the centre, rotating 180 degrees and then moving back.

ii) I see no way this can be done in one transformation, as each individual “line” of the figure needs the be calculated separately. By using some sort of Pythagoras on the line and sphere theorem this might be possible, but still not with only one transformation.

iii) This can be done through projection transformation

iv) Can’t be done. The reason is that only moving one object while having the other unaffected is impossible with one transformation

v) This can be done, though we need to modify our regular shear transformation a little. The reason for this is the exponential growth of the y-axis the further down the x-axis it is. So to accomplish this, I suggest shearing, not a constant of c*x, but rather with e^cx. In addition, we need to translate it a little to the left before we do the “shearing” and then translating it back, as our modified image has a lesser hight in the beginning of the figure compared to the original.

The results of my program:

In the following section I have put screenshots of my results.

Task 1)

The first picture is of the 5 triangles made up in total by 15 vertices, all with different colors. The data for this is represented in the next two pictures.

Task 4:

i) The following 4, the ones with the small pyramid, is my final program. In these pictures I have used the perspective projection. This works wonders, as I can move freely around without anything getting clipped. In these shots I:

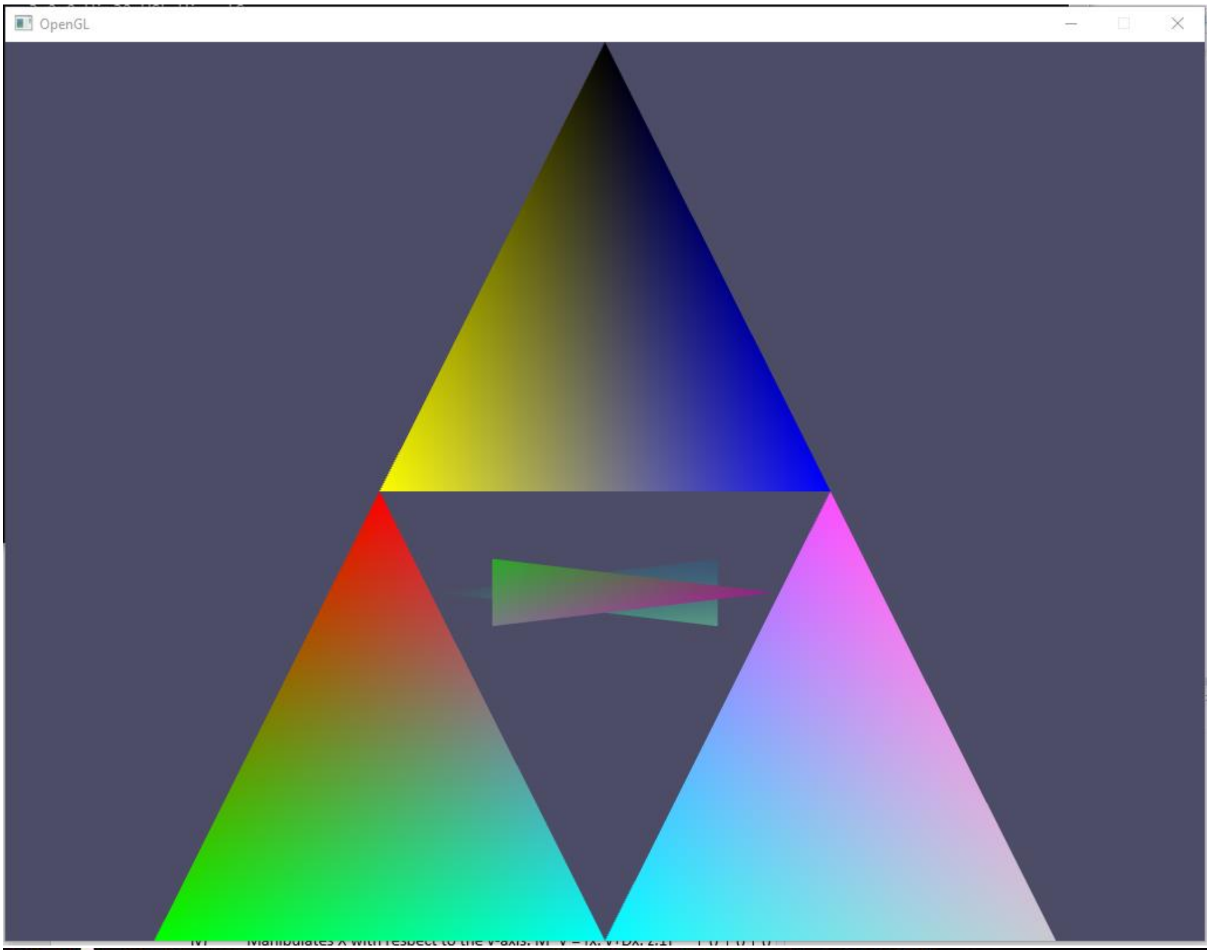
1. Start the program and move backwards to get the full pyramid
2. I rotate it slightly and move sideways to get the edges.
3. From here I move down to see the underside of it, looking up
4. From here I move to see it from the pointy blue side.

ii) The last 4 pictures are the ones with the program implemented with the orthraphic projection. I had set the projection to exaclty fit my cordinates. This caused some of the vertices to getting clipped when moving and rotating too much.
In these shots:

1. Start the program
2. Rotating, causing the foremost edge to getting clipped.
3. Back to start, rotating upwards causing the tip to ble clipped
4. Moving in such a manner that the whole pyramid is still inside the clip-box.

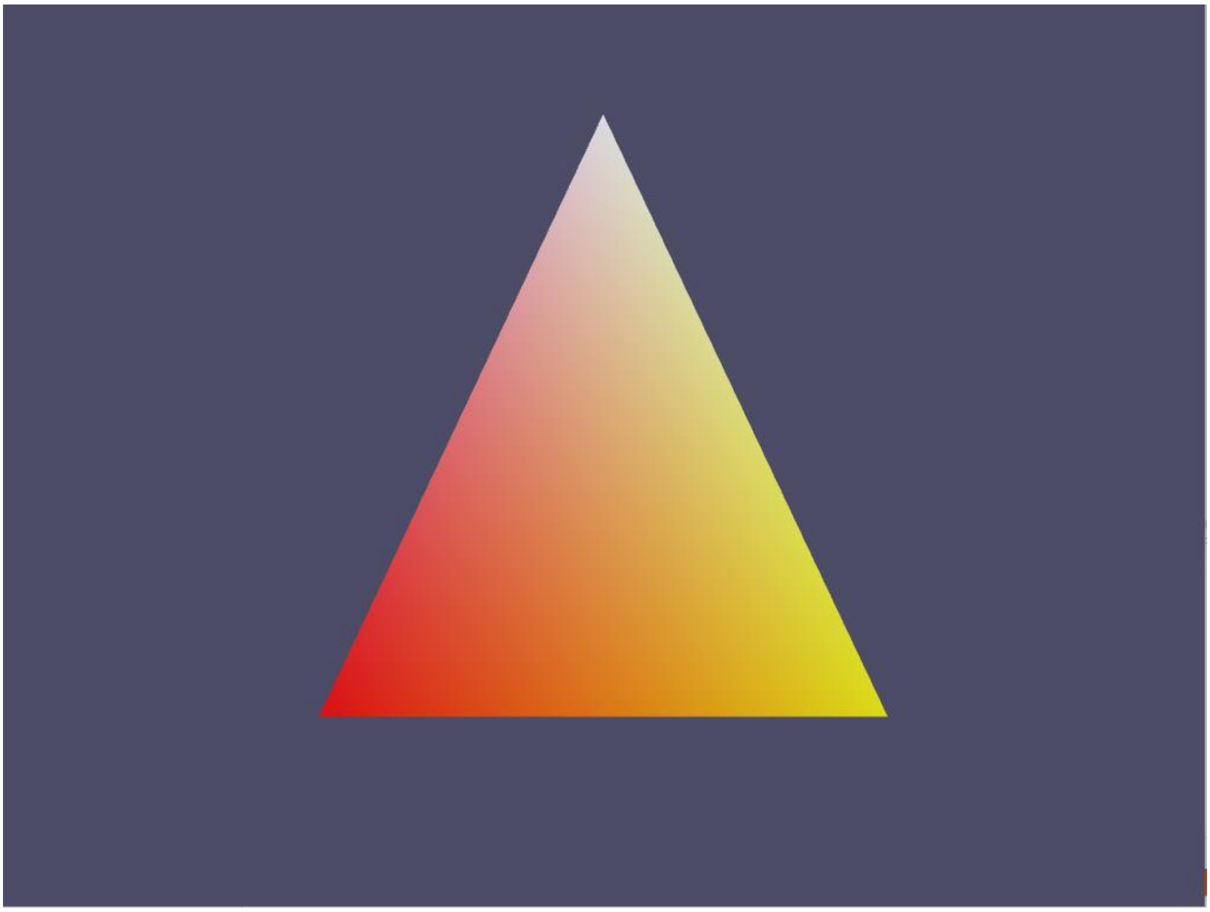
In my program: moving the camera around in the xy-plane is done with the key-arrows. Rotating the camera is done with WASD. Moving the camera forwards and backwards is done with Enter and Backspace.

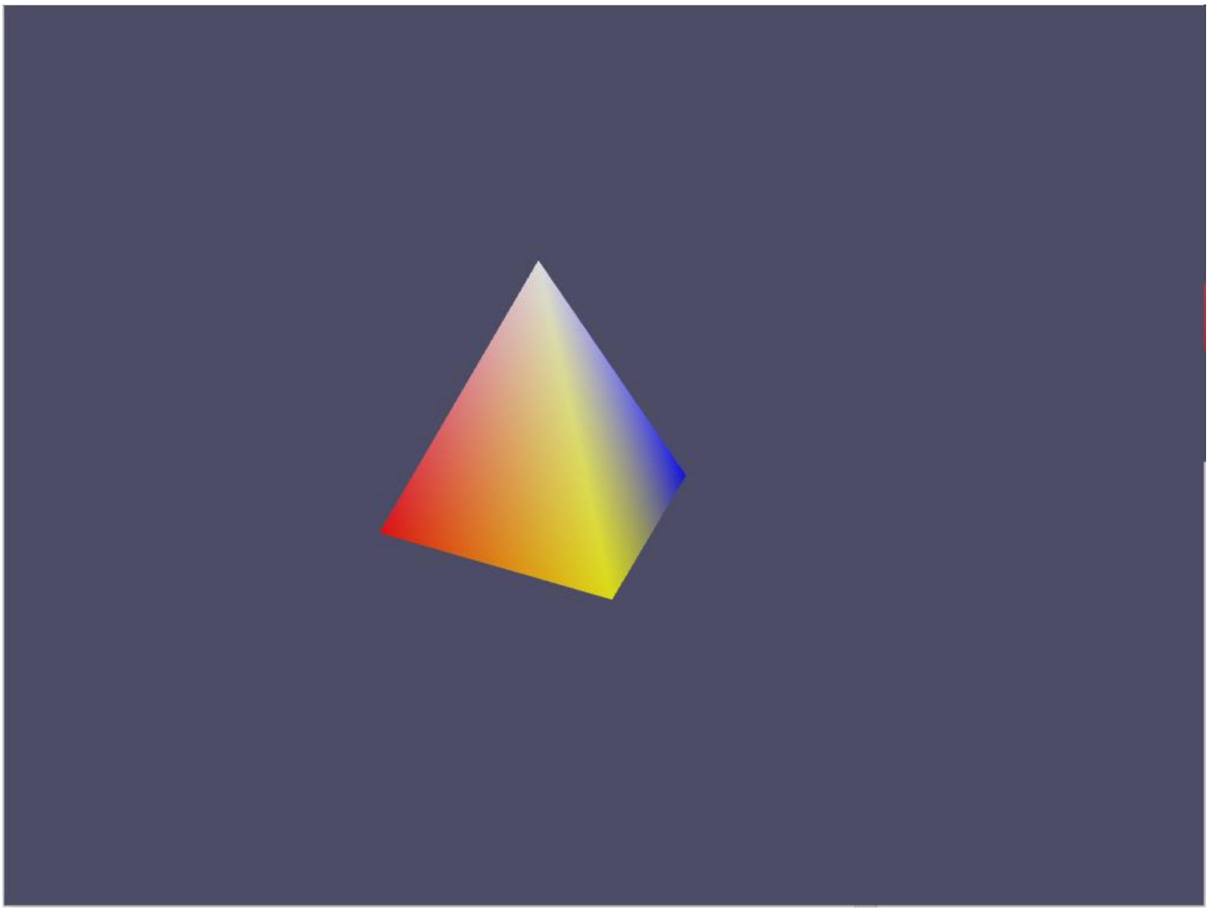
In addition, I added some of the algorithms used. The first one is the extended function taking in the colors of the vertices, and the second one is the matrix for calculating the view matrix and updating the MVP.

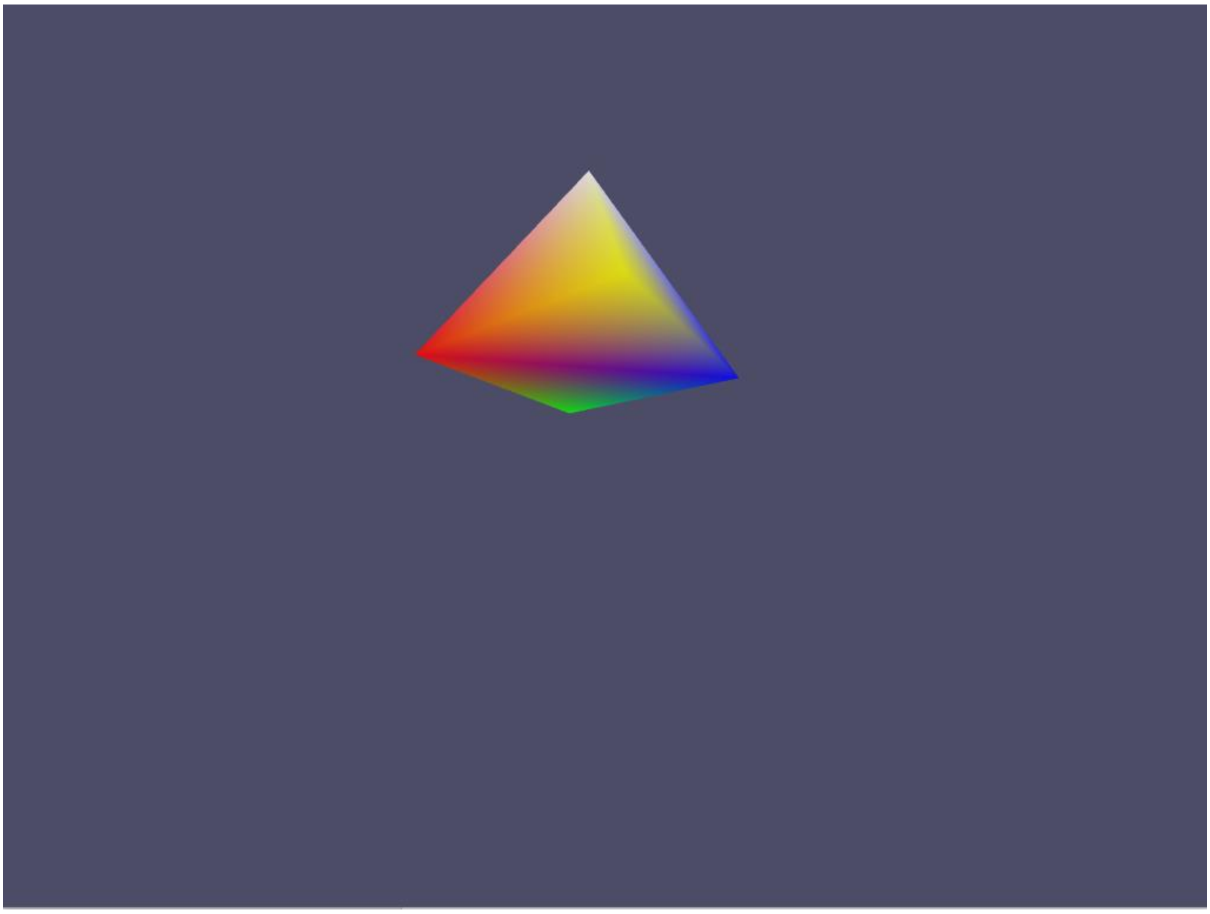


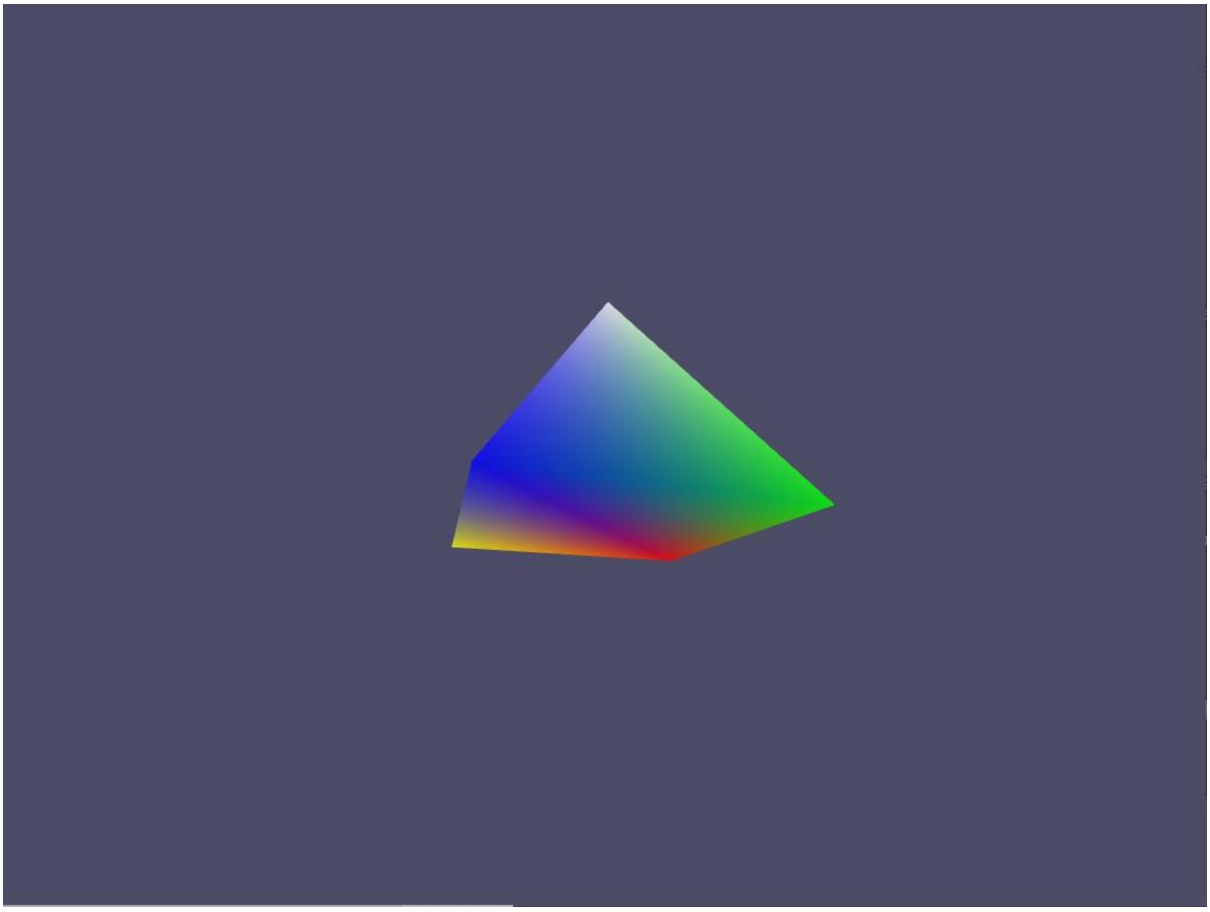

```
void build_triangles(unsigned int& array,unsigned int& num_indices){
    float vertices[]={
      0,1,0,
      -0.5,0,0,
      0.5,0,0,

      -0.5,0,0,
     -1,-1,0,
      0,-1,0,

      0.5,0,0,
      0,-1,0,
      1,-1,0,

      -0.25,-0.15,0,
      -0.25,-0.30,0,
      0.375,-0.225,0,

      0.25,-0.15,0,
      0.25,-0.30,0,
      -0.375,-0.225,0
    };
    projectionMatrix = glm::ortho(-1,1,-1,1,-1,1);


    int indices[] = {
      0,1,2,
      3,4,5,
      6,7,8,
      9,10,11,
      12,14,13 //had to flip to make it show, hence 14 and 13 in that order
    };

    float colors[] = {
        0,0,0,1,
```

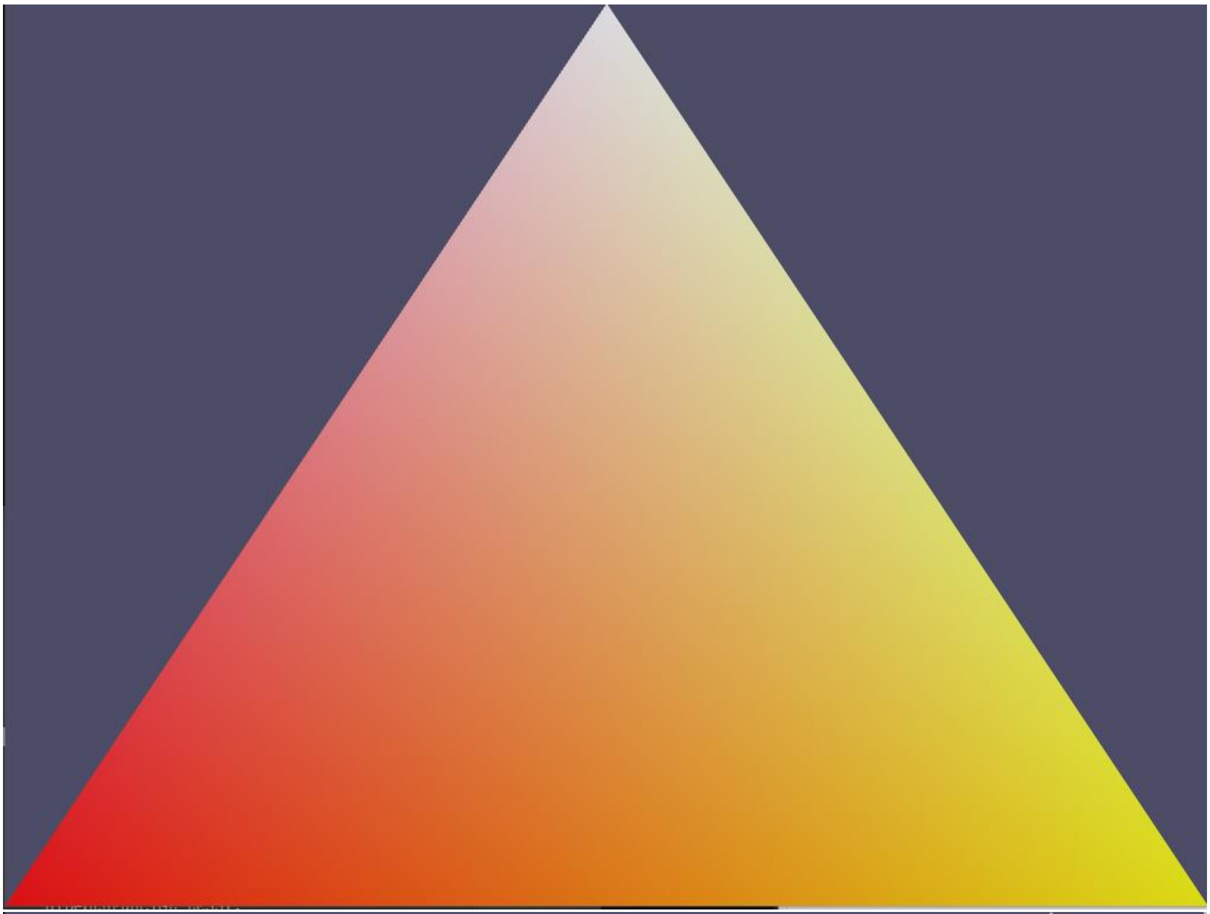
```
    float colors[] = {
        0,0,0,1,
        1,1,0,1,
        0,0,1,1,

        1,0,0,1,
        0,1,0,1,
        0,1,1,1,

        1,0.3,1,
        1,0,1,1,
        1,1,1,1,

        0.7,  0.1,  0.9,  0,
        0.6,  0.6,  0.6,  0.6,
        0.6,  0.8,  0,    0.6,

        0.6,  0.1,  0.4,  0.6,
        0.3,  0.4,  0.8,  0.6,
        0.6,  0,    1,    0.6

    };
```

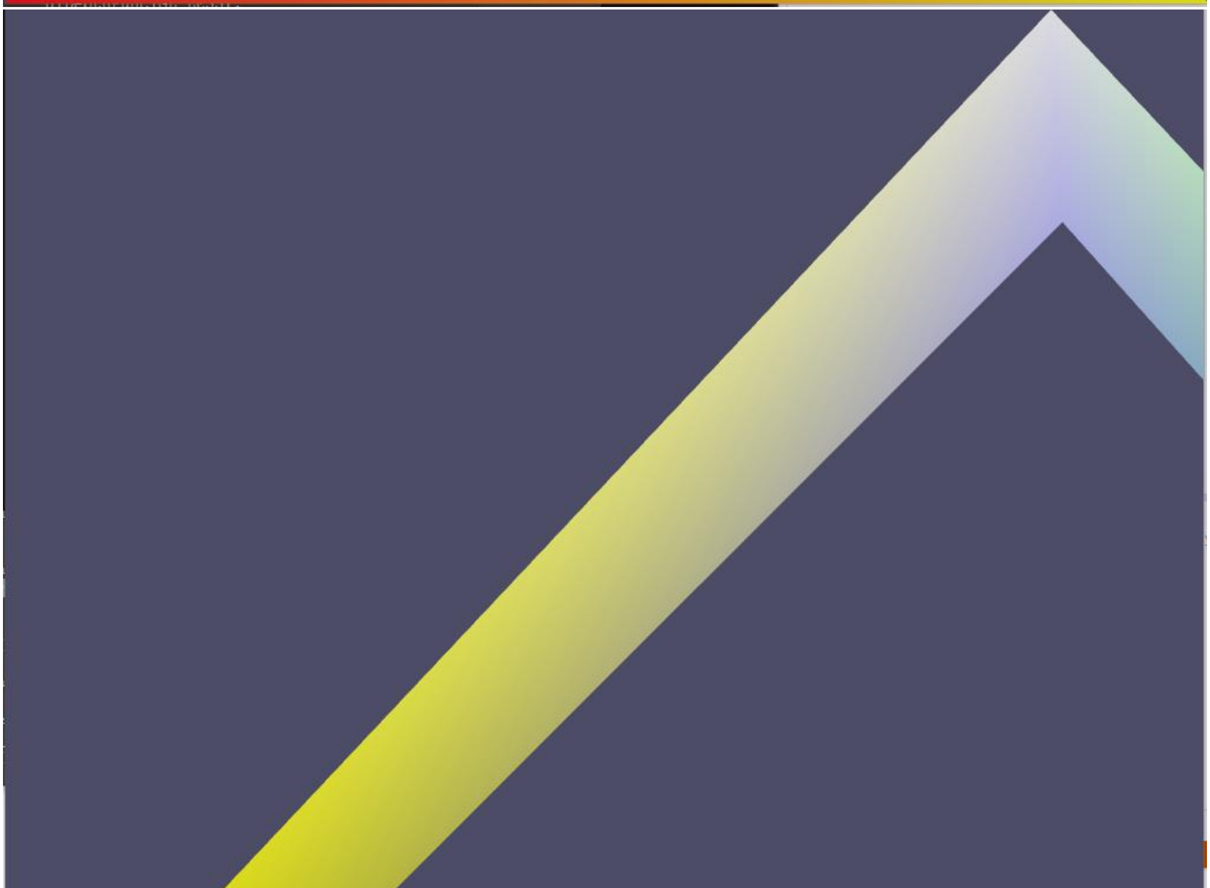


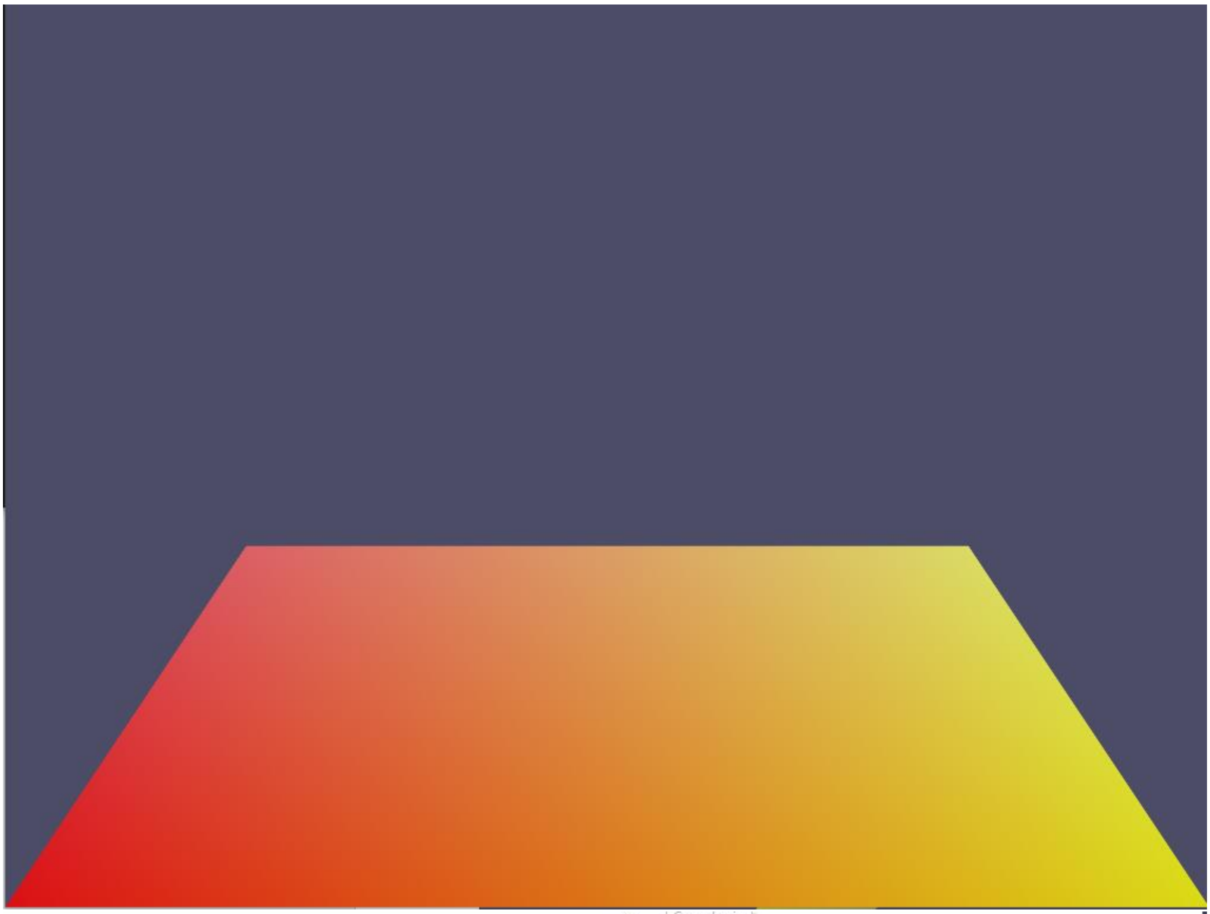


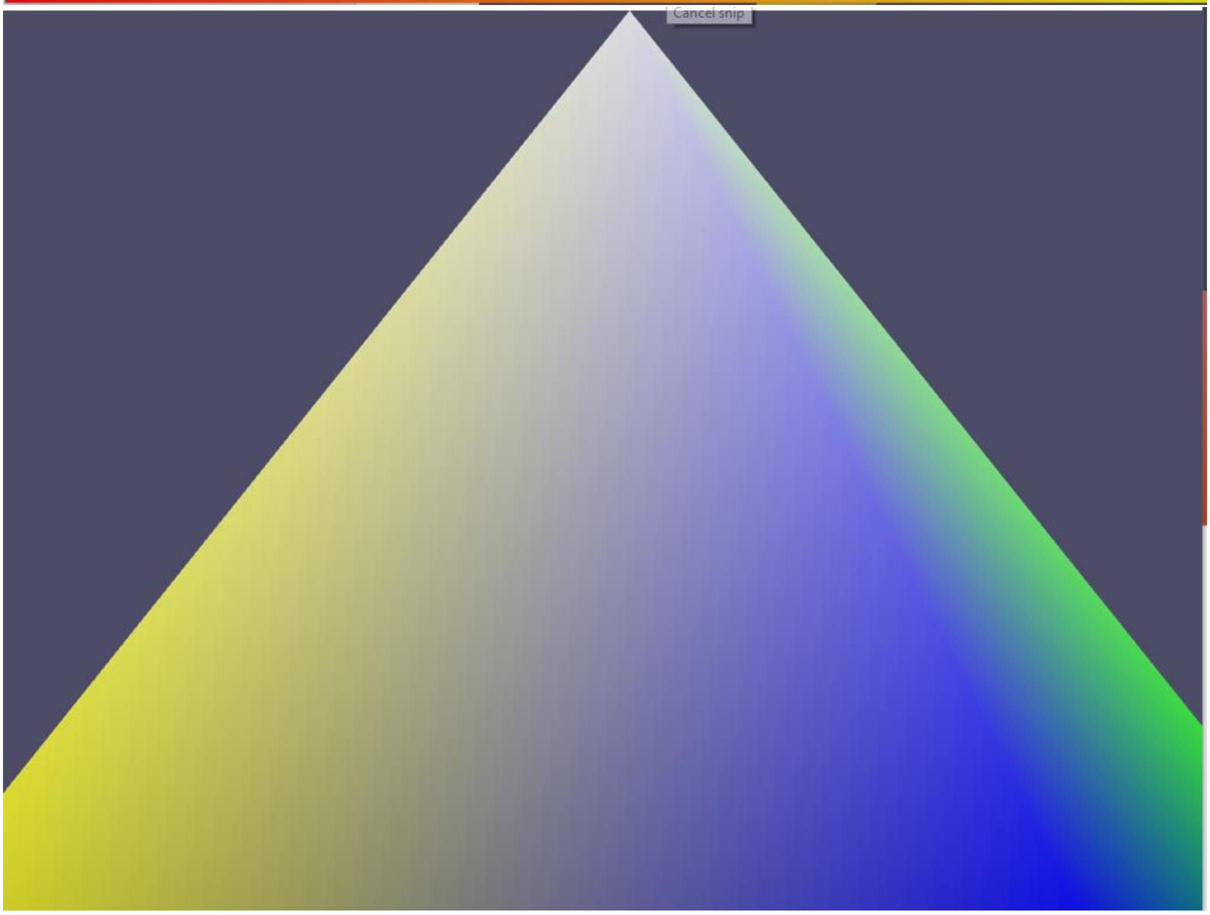











Cancel snip

```
unsigned int setupVBO(float* vertices,int vertices_len,int* indices,int indices_len,float* color
s,int colors_len){
    //Creates and binds VAO, referenced in array
    GLuint arrayID;
    glGenVertexArrays(1,&arrayID);
    glBindVertexArray(arrayID);

    //creates buffer and binds it
    GLuint vertexID;
    glGenBuffers(1,&vertexID);
    glBindBuffer(GL_ARRAY_BUFFER,vertexID);

    //Finds vertice lengthm and pushes the data to the buffer
    glBufferData(GL_ARRAY_BUFFER,vertices_len,vertices,GL_STATIC_DRAW);

    //sets Vertext Attribute Pointer and enables the VBO
    glVertexAttribPointer(0,3,GL_FLOAT,GL_FALSE,0,0);
    glEnableVertexAttribArray(0);

    //creates the Index Buffer
    GLuint indicesID;
    glGenBuffers(1,&indicesID);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,indicesID);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,indices_len,indices,GL_STATIC_DRAW);

    //create colors
    GLuint colorID;
    glGenBuffers(1,&colorID);
    glBindBuffer(GL_ARRAY_BUFFER,colorID);
    glBufferData(GL_ARRAY_BUFFER,colors_len,colors,GL_STATIC_DRAW);

    //sets pointer and enables
    glVertexAttribPointer(1,4,GL_FLOAT,GL_FALSE,0,0);
    glEnableVertexAttribArray(1);

    return arrayID;
}
```

```
void updateMVP() {
  //direction of current view
  glm::vec4 direction(orientation);

  //rotates camera to x=0 by revolving around y-axis, formula found on internet
  glm::mat4x4 rotation_x = glm::rotate(atan2(-direction.x, direction.z), y_axis);
  direction = rotation_x * direction;

  //rotates camera to y=0 by revoling around x-axis with formula found on internet
  glm::mat4x4 rotation_y = glm::rotate(atan(direction.y/ direction.z), x_axis);
  direction = rotation_y *direction;

  //rotate camera to z=-1 by revolving around y-axis if its set to z=1
  glm::mat4x4 rotation_z = glm::mat4(1.0);
  if (direction.z > 0)
    rotation_z = glm::rotate(pi, y_axis);
  direction = rotation_z * direction;

  //compute full rotation transformation
  glm::mat4x4 rotation_transformation = rotation_z * rotation_y * rotation_x;

  //finds current up_vector of this transformation
  glm::vec4 up = rotation_transformation * up_vector;

  //finds transformation that rotates to "up"
  glm::mat4x4 rotate_Up = glm::rotate(atan2(up.x, up.y), z_axis);
  //rotates to up
  rotation_transformation = rotate_Up * rotation_transformation;

  // Tranformation for moving to eye view
  glm::vec3 eye_view = -glm::vec3(position);
  //computes full transformation from rotation and eye-view transformations
  glm::mat4x4 viewMatrix = rotation_transformation * glm::translate(eye_view);


  //MVP = viewMatrix;
  MVP = projectionMatrix * viewMatrix * modelMatrix;
}
```