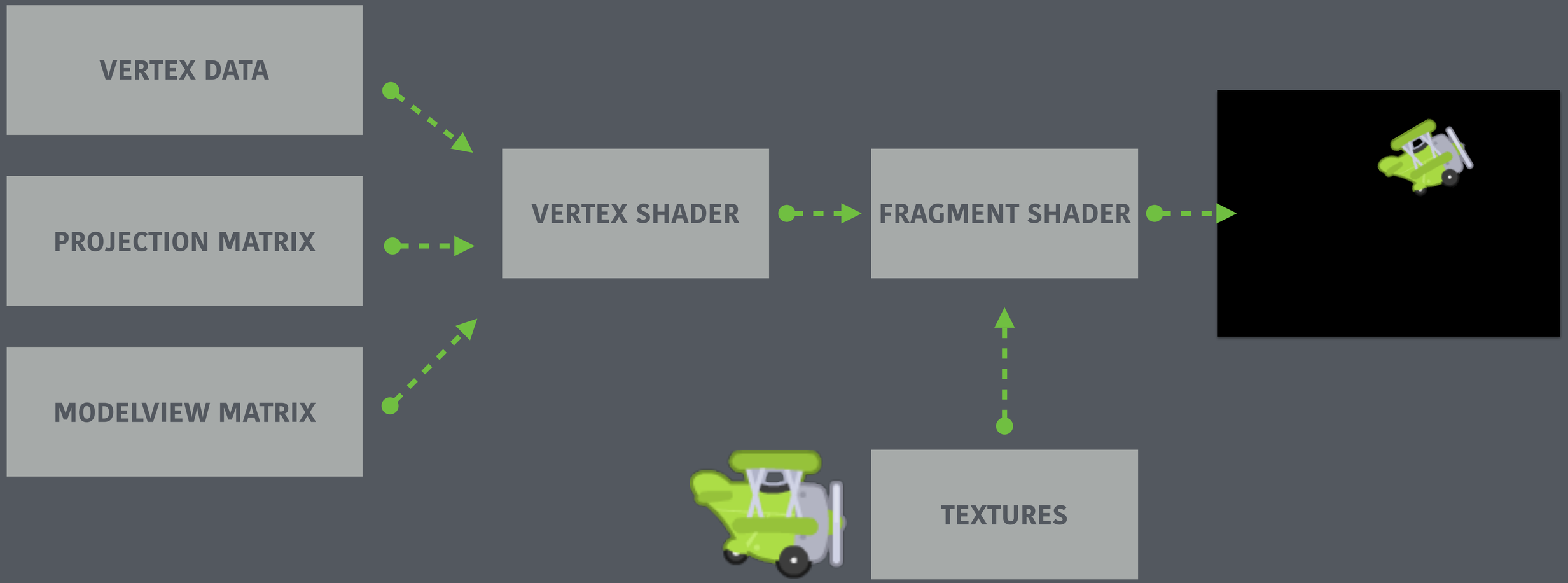
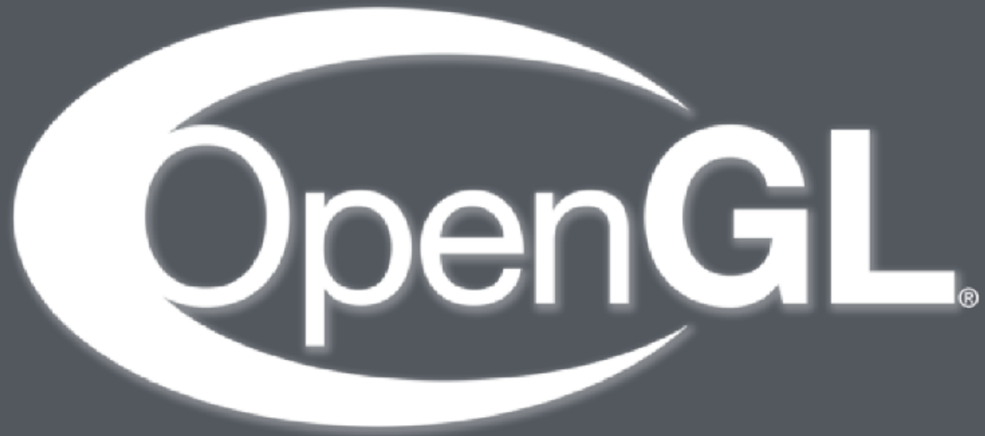
Graphics Foundations

Part 2

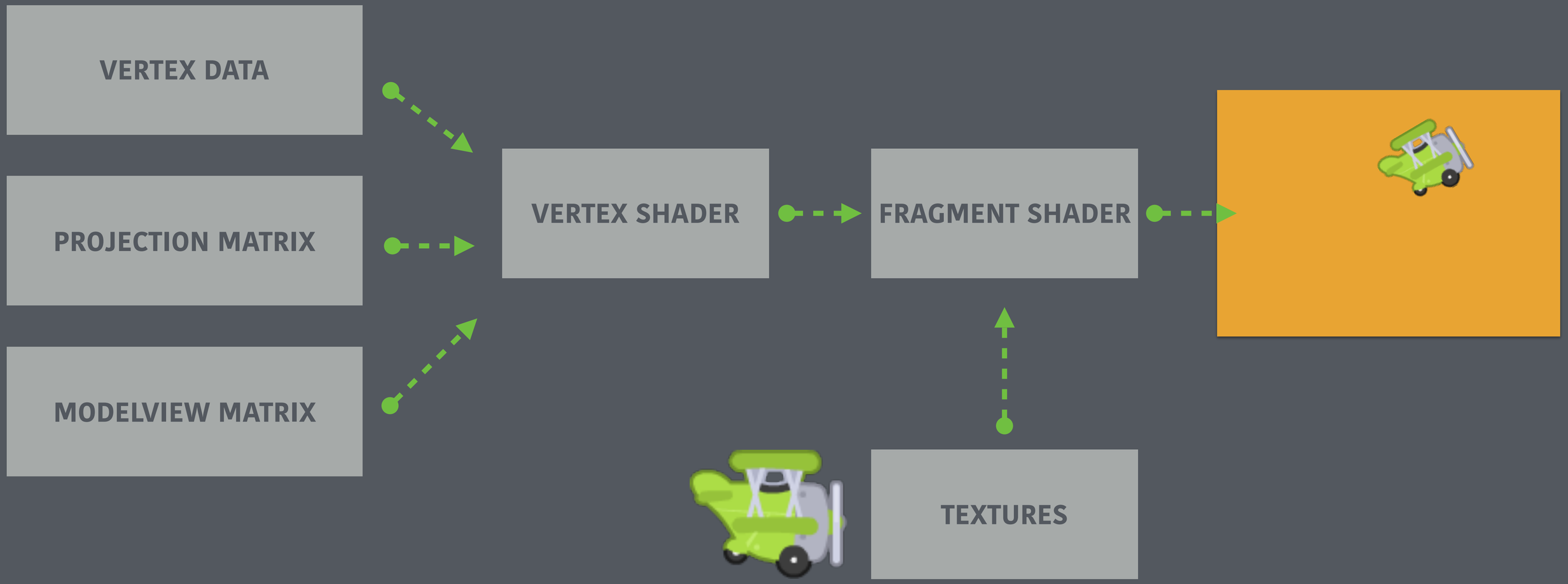
The **GPU** pipeline




OpenGL®

The setup
(happens only once!)

The **GPU** pipeline

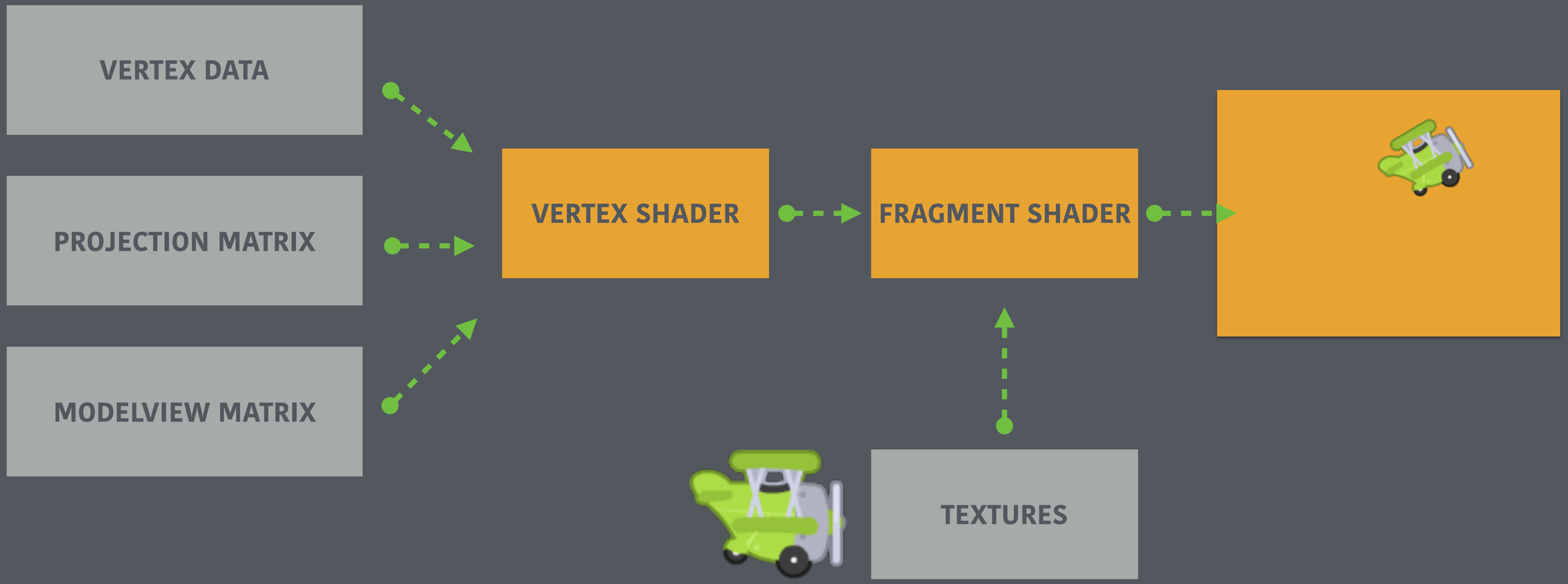


```
void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);
```

Sets the size and offset of rendering area (in pixels).

```
glViewport(0, 0, 640, 360);
```

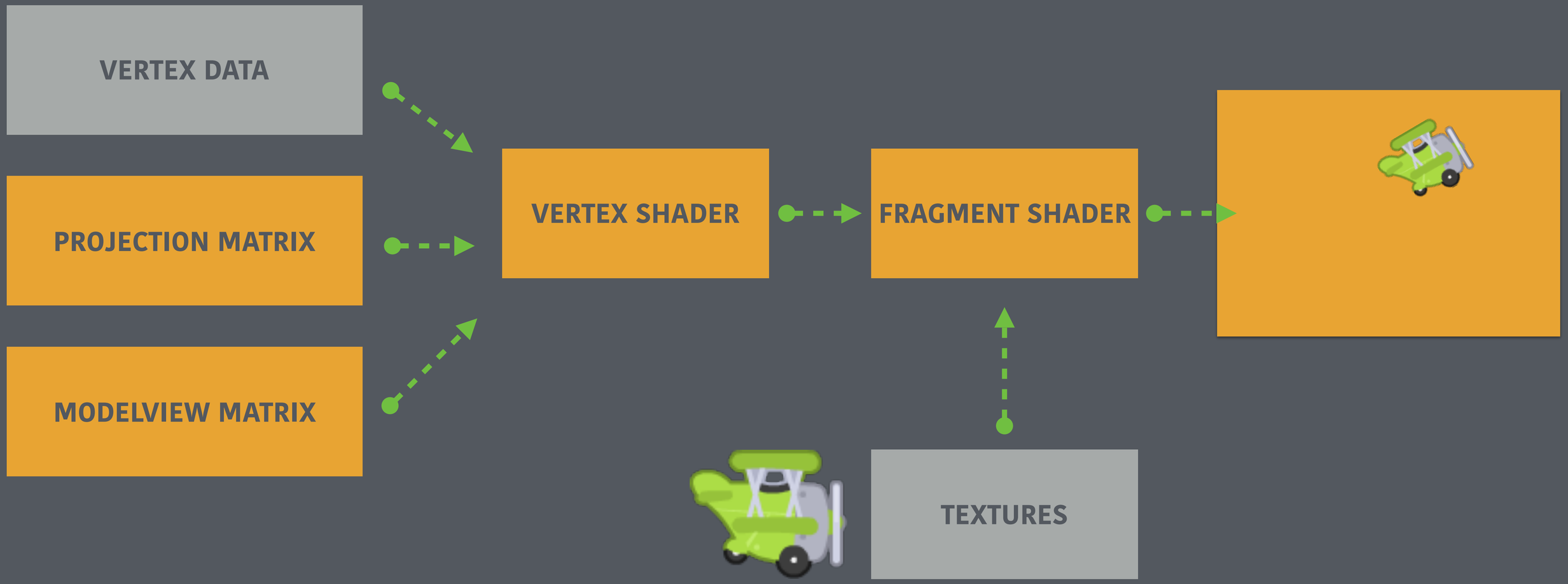
The **GPU** pipeline



The **ShaderProgram** class.

```
#include "ShaderProgram.h"

// …

ShaderProgram program(RESOURCE_FOLDER"vertex.glsl", RESOURCE_FOLDER"fragment.glsl");
```

The **GPU** pipeline



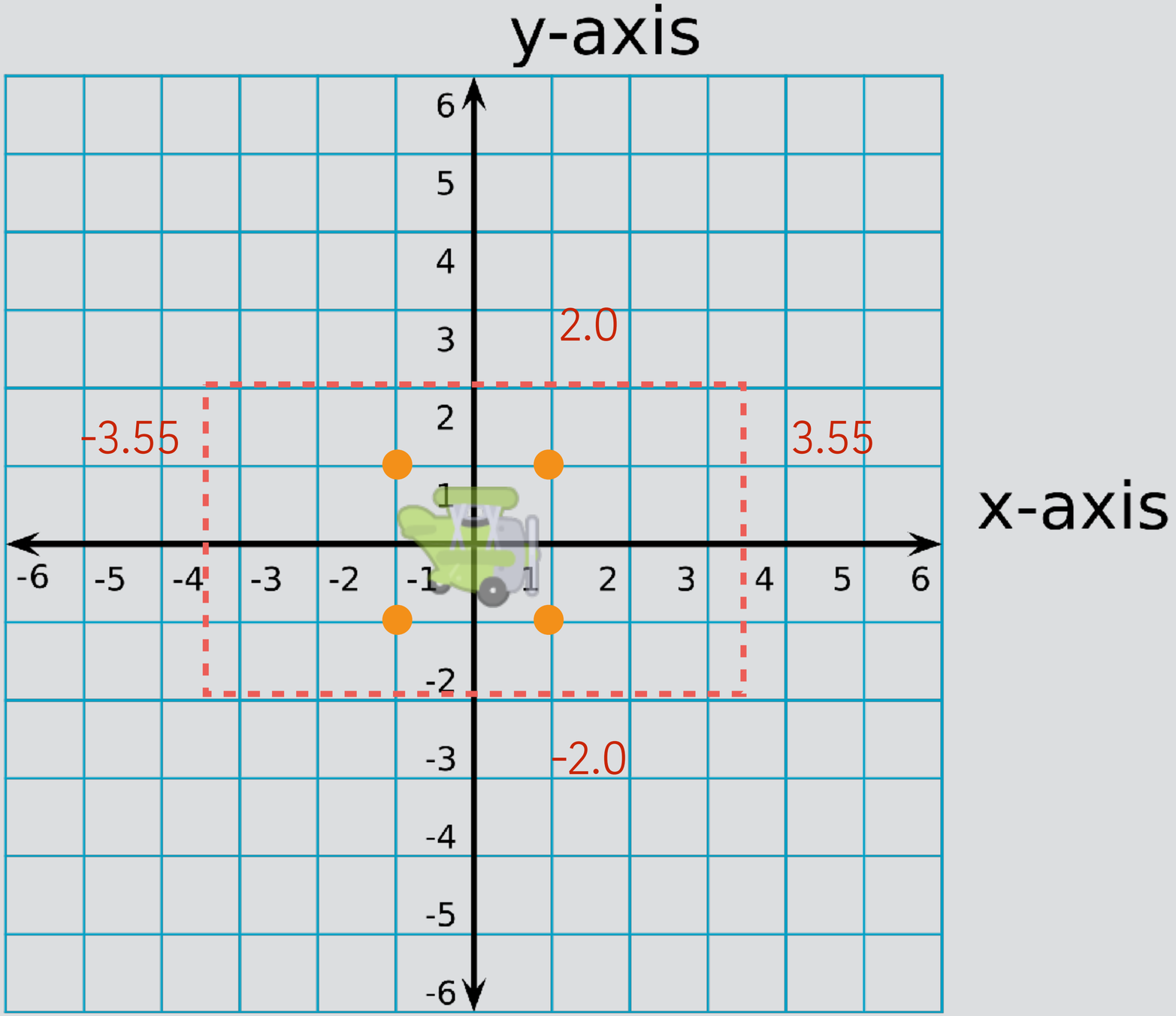
The **Matrix** class.

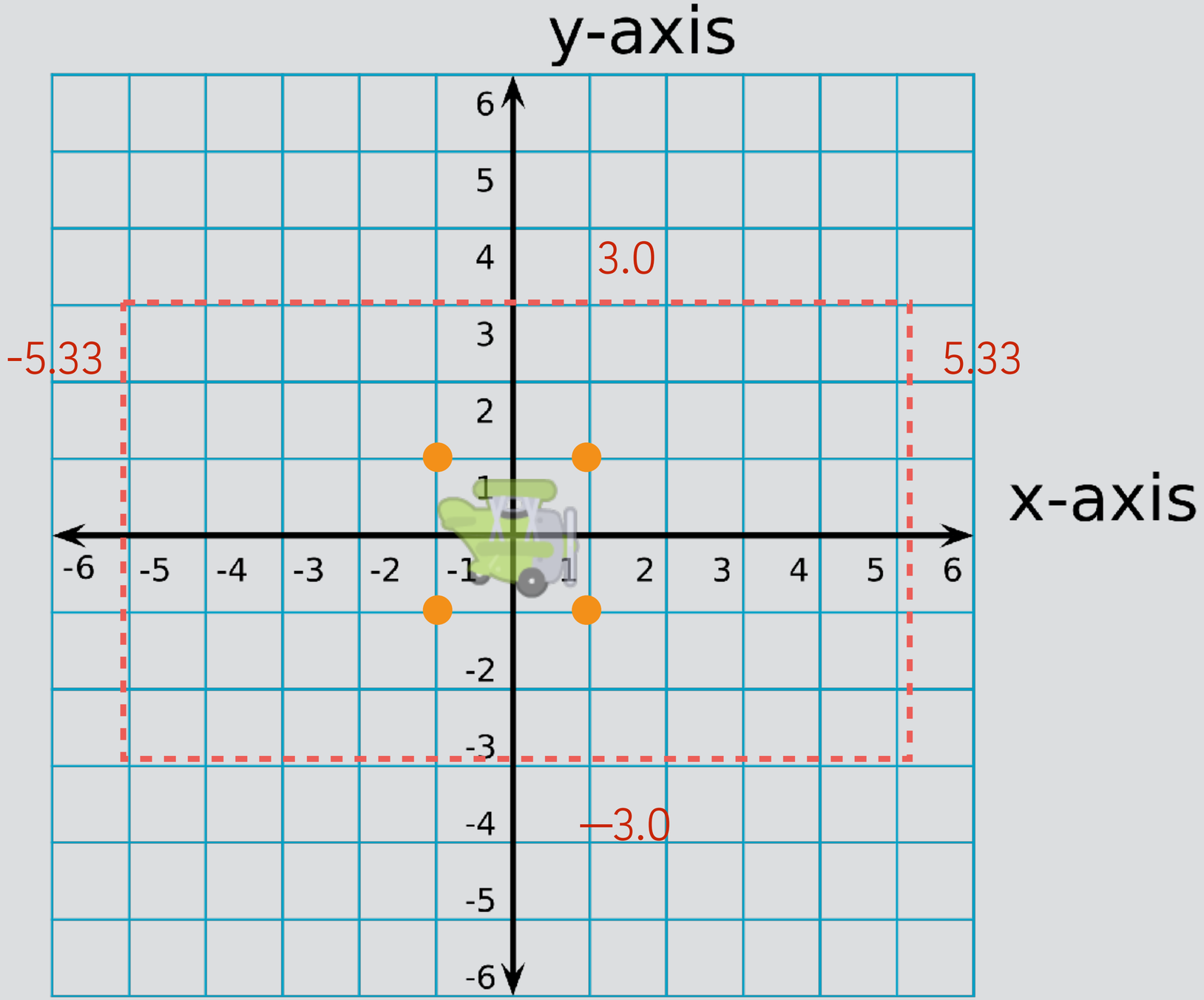
```
#include "Matrix.h"

// …

Matrix projectionMatrix;
Matrix modelviewMatrix;
```

```
void Matrix::SetOrthoProjection (float left, float right,
float bottom, float top, float near, float far);
```

Sets an orthographic projection in a matrix.

```
Matrix projectionMatrix;
projectionMatrix.SetOrthoProjection(-3.55f, 3.55f, -2.0f, 2.0f, -1.0f, 1.0f);
```


y-axis
x-axis
2.0
-2.0
-3.55
3.55
6
5
4
3
2
1
-1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6


y-axis
x-axis
3.0
-3.0
-5.33
5.33
6
5
4
3
2
1
-1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

Drawing polygons.
(happens every frame!)

`void glUseProgram (GLint programID);`

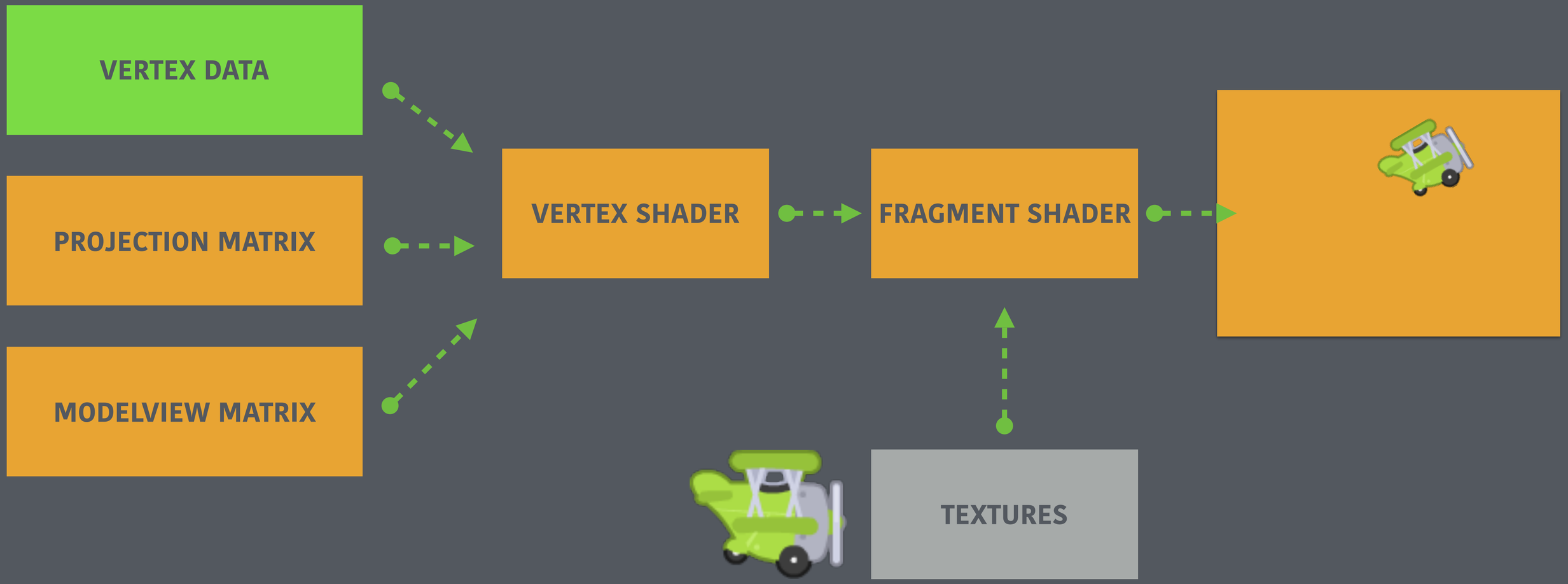
Use the specified **program id**. We need to tell OpenGL what shader program to use before we render anything!

```
glUseProgram(program.programID);
```

Pass the **matrices** to our **program**.

```
program.SetProjectionMatrix(projectionMatrix);
program.SetModelviewMatrix(viewMatrix);
```

The **GPU** pipeline



```
void glVertexAttribPointer (GLint index, GLint
size, GLenum type, GLboolean normalized, GLsizei
stride, const GLvoid *pointer);
```

Defines an array of **vertex data** (**counter**-clockwise!).

```
float vertices[] = {0.5f, -0.5f, 0.0f, 0.5f, -0.5f, -0.5f};

glVertexAttribPointer(program.positionAttribute, 2, GL_FLOAT, false, 0, vertices);
```

```
void glEnableVertexAttribArray (GLuint index);
```

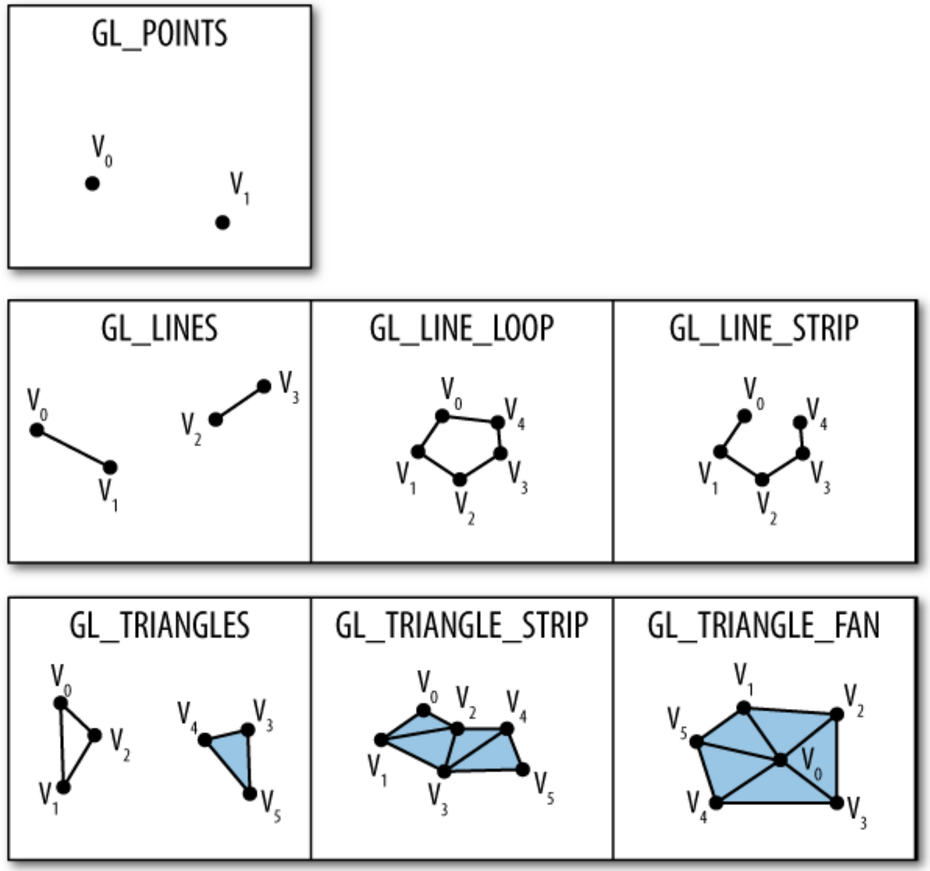
Enables a **vertex attribute**.

```
glEnableVertexAttribArray(program.positionAttribute);
```

```
glDrawArrays (GLenum mode, GLint first,
GLsizei count);
```

Render previously defined **arrays**.

```
glDrawArrays(GL_TRIANGLES, 0, 3);
```



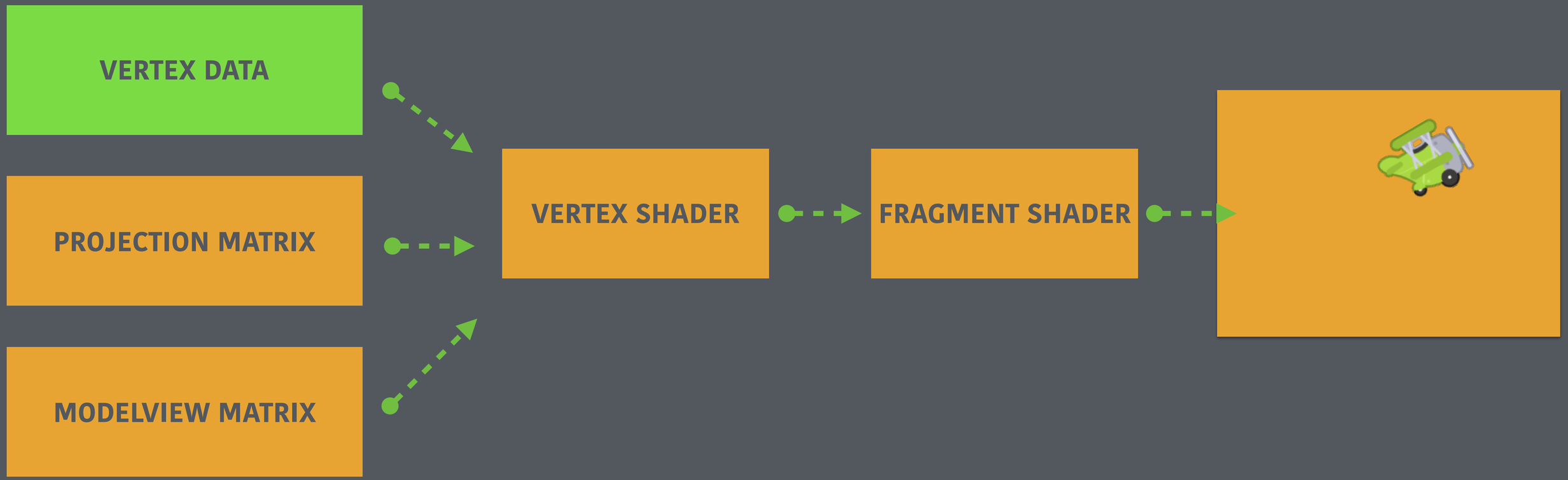
```
void glDisableVertexAttribArray (GLuint index);
```

Disables a **vertex attribute array**.

```
glDisableVertexAttribArray(program.positionAttribute);
```

Putting it all together.

The **GPU** pipeline



Setup (before the loop)

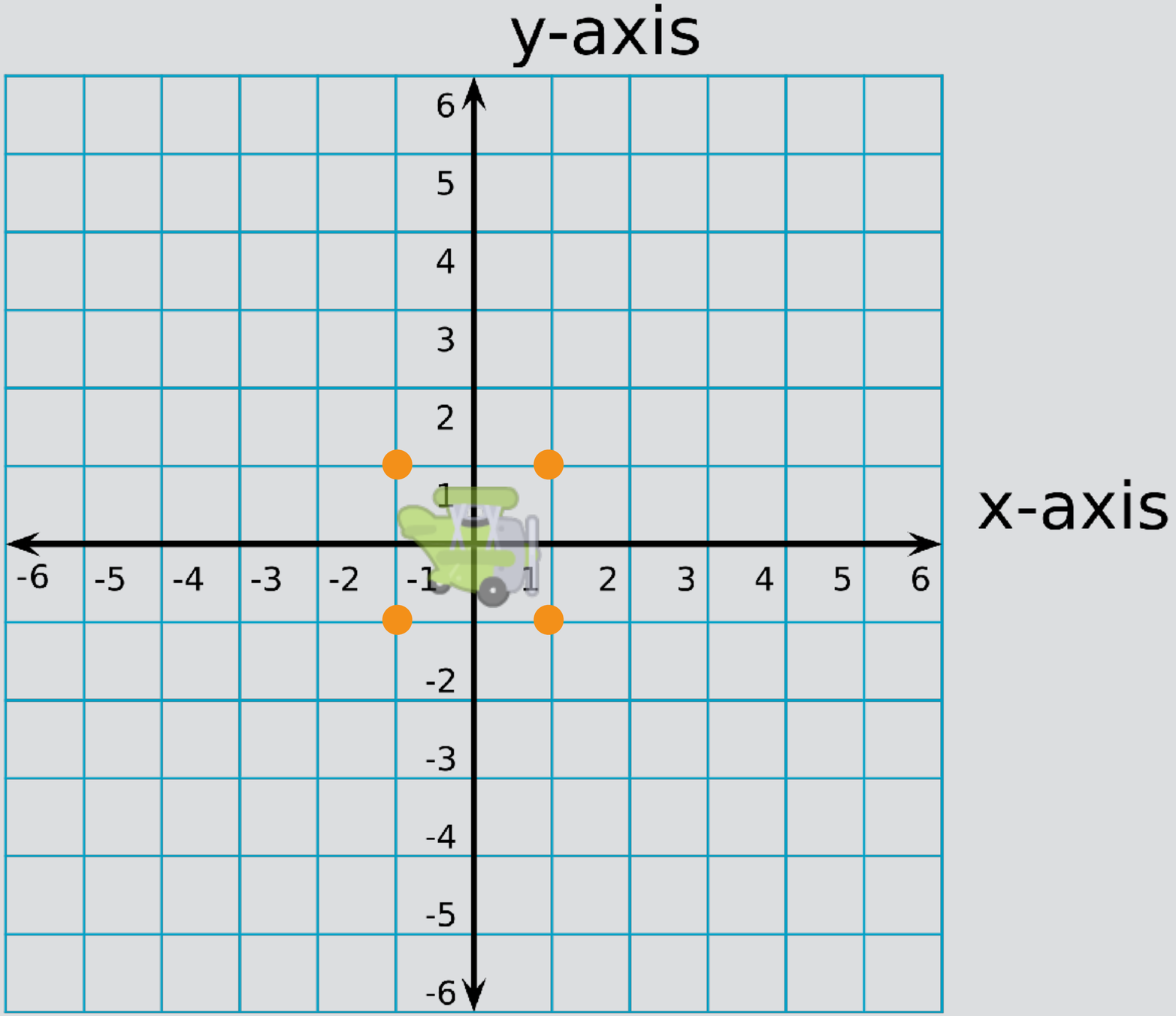
```
glViewport(0, 0, 640, 360);

ShaderProgram program(RESOURCE_FOLDER"vertex.glsl", RESOURCE_FOLDER”fragment.glsl");

Matrix projectionMatrix;
projectionMatrix.SetOrthoProjection(-3.55f, 3.55f, -2.0f, 2.0f, -1.0f, 1.0f);
Matrix modelviewMatrix;
```

Drawing (in your game loop)

```
glClear(GL_COLOR_BUFFER_BIT);

glUseProgram(program.programID);

program.SetModelviewMatrix(modelviewMatrix);
program.SetProjectionMatrix(projectionMatrix);

float vertices[] = {0.5f, -0.5f, 0.0f, 0.5f, -0.5f, -0.5f};
glVertexAttribPointer(program.positionAttribute, 2, GL_FLOAT, false, 0, vertices);
glEnableVertexAttribArray(program.positionAttribute);

glDrawArrays(GL_TRIANGLES, 0, 3);

glDisableVertexAttribArray(program.positionAttribute);

SDL_GL_SwapWindow(displayWindow);
```

Transformations

Identity matrix.

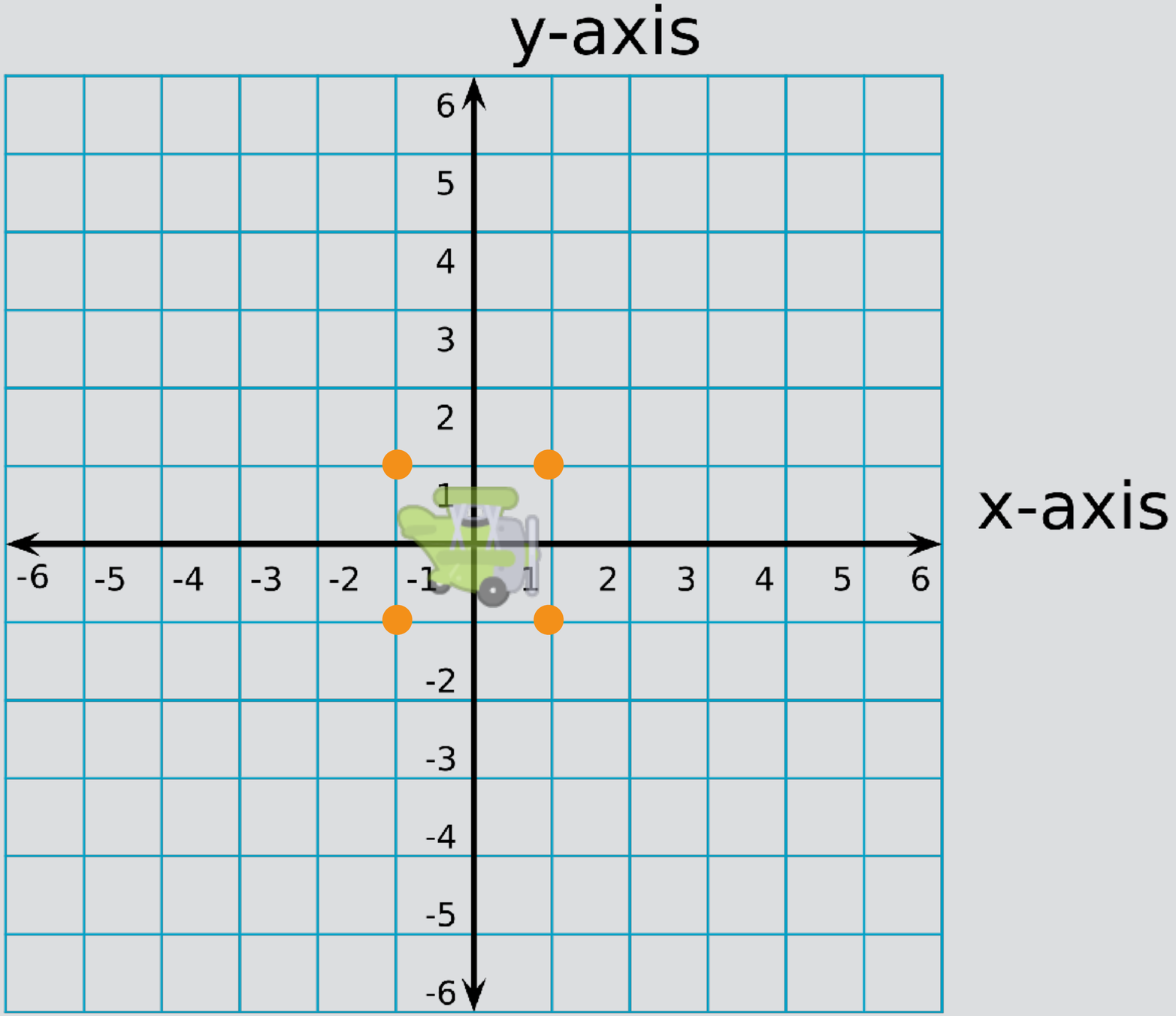

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

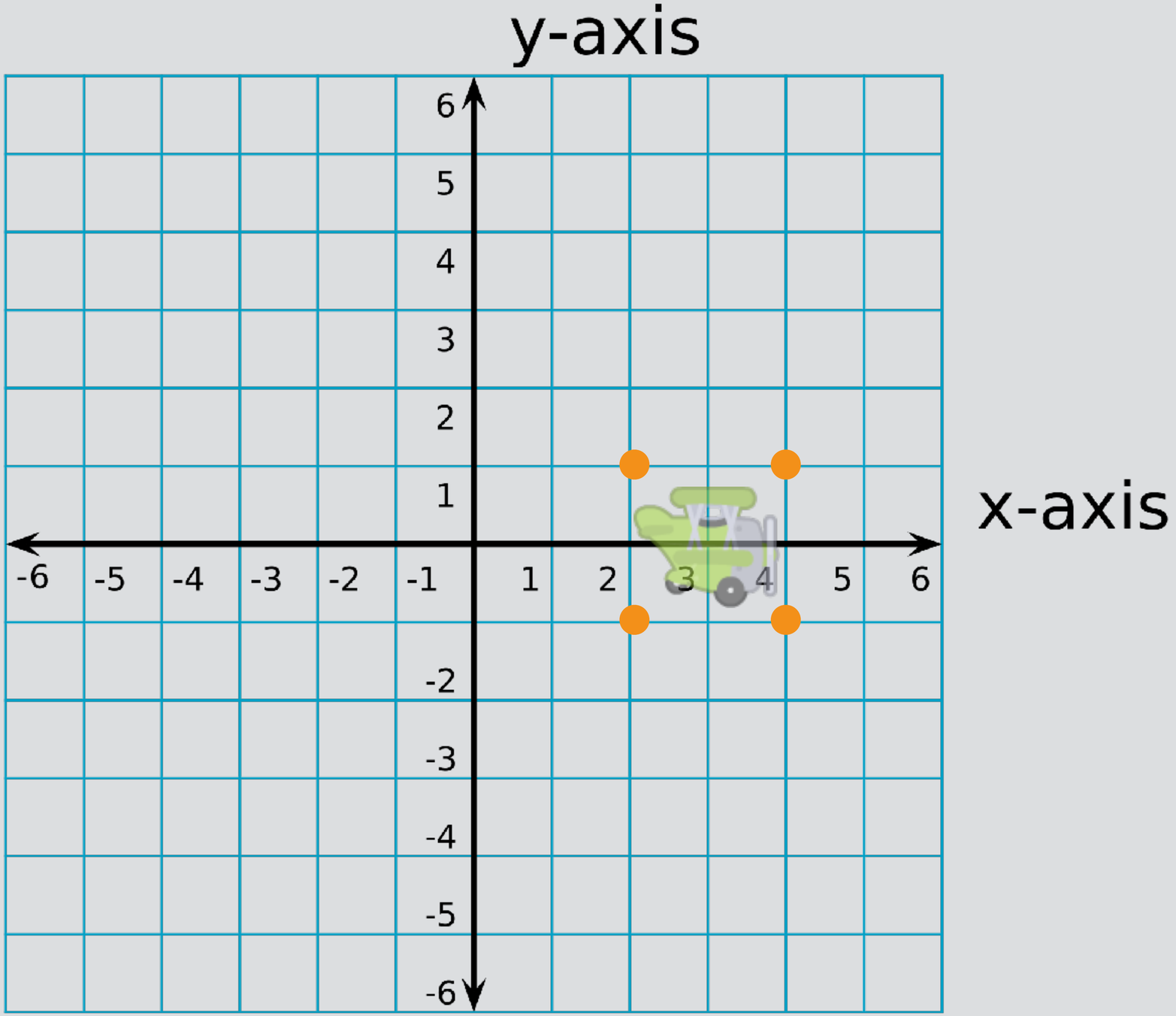
```
void Matrix::Identity();
```

Resets a matrix to have no transformations.

```
Matrix modelviewMatrix;
modelviewMatrix.Identity();
```

Translations


y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

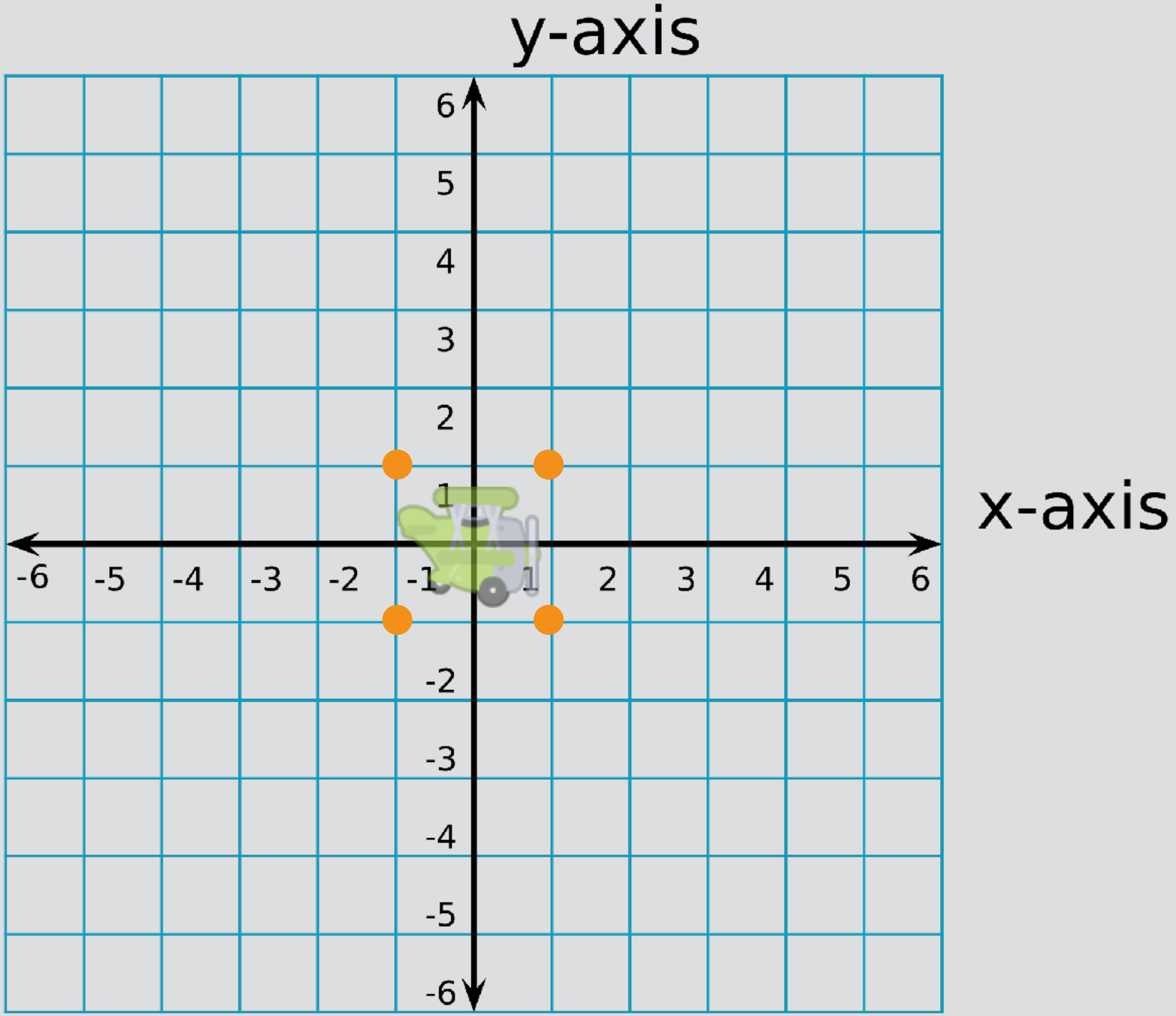

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

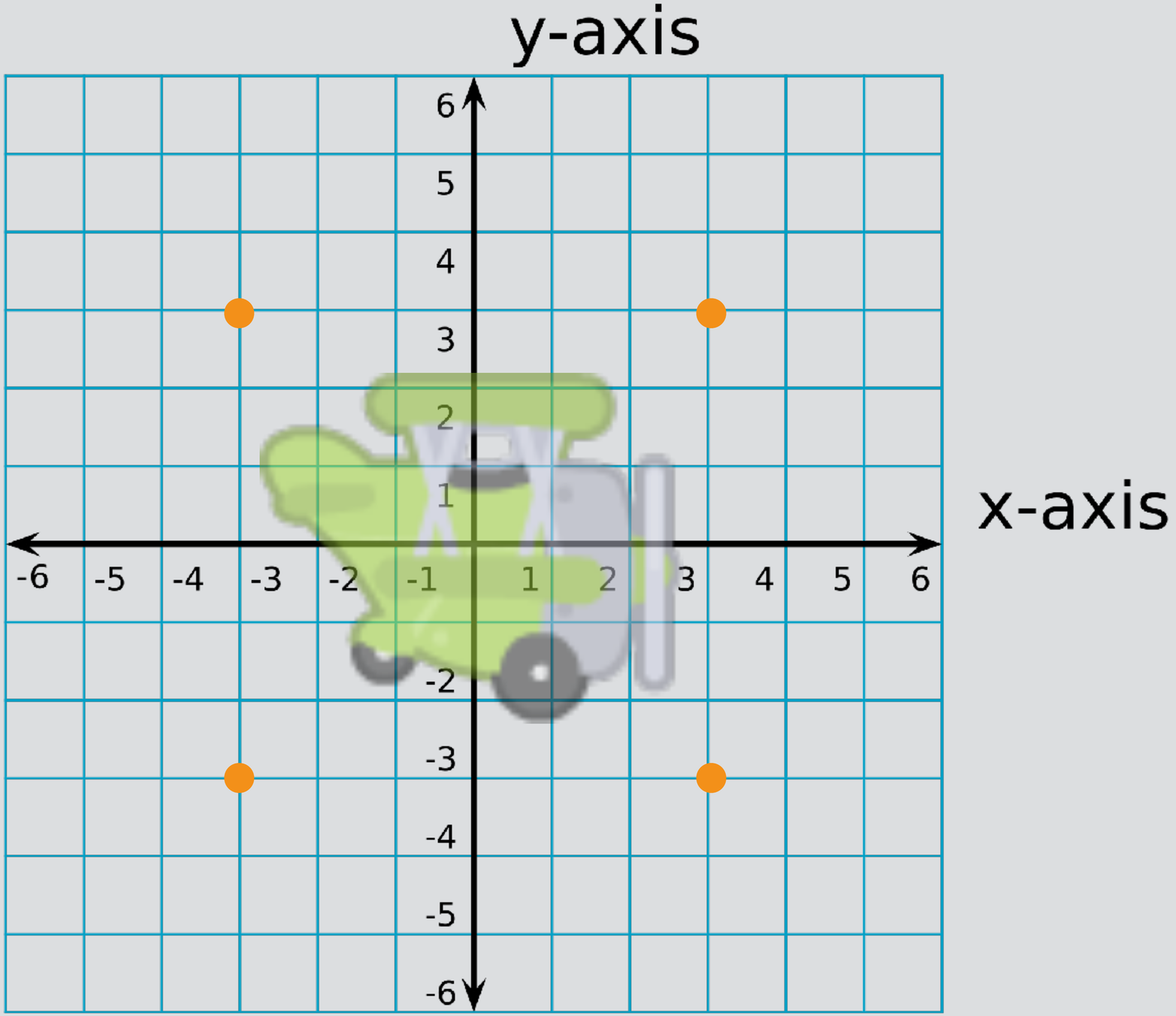
```
void Matrix::Translate (float x, float y, float z);
```

Translates a matrix by specified coordinates.

```
Matrix modelviewMatrix;
modelviewMatrix.Translate(2.0f, 0.0, 0.0);
```

Scaling


y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

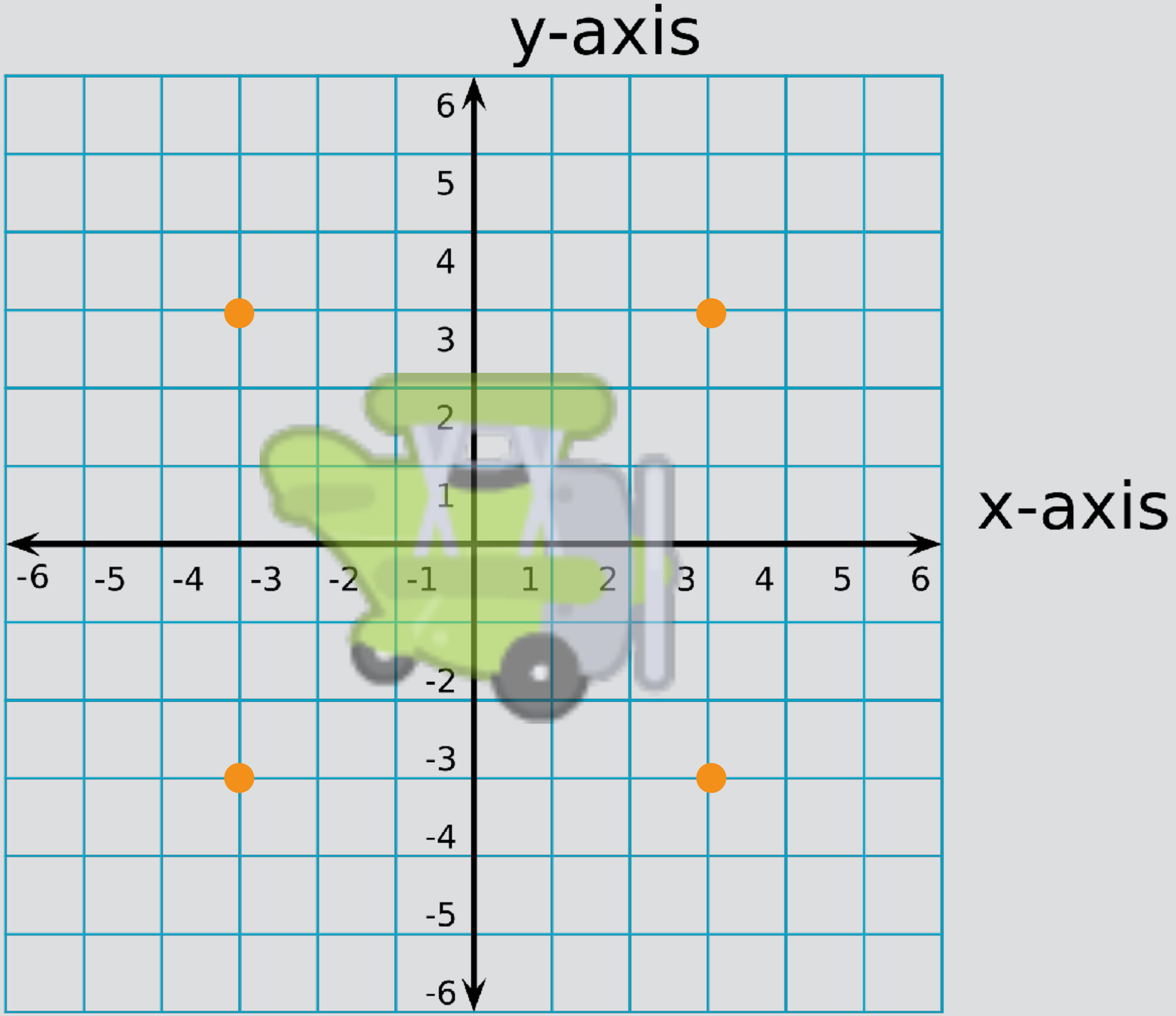

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

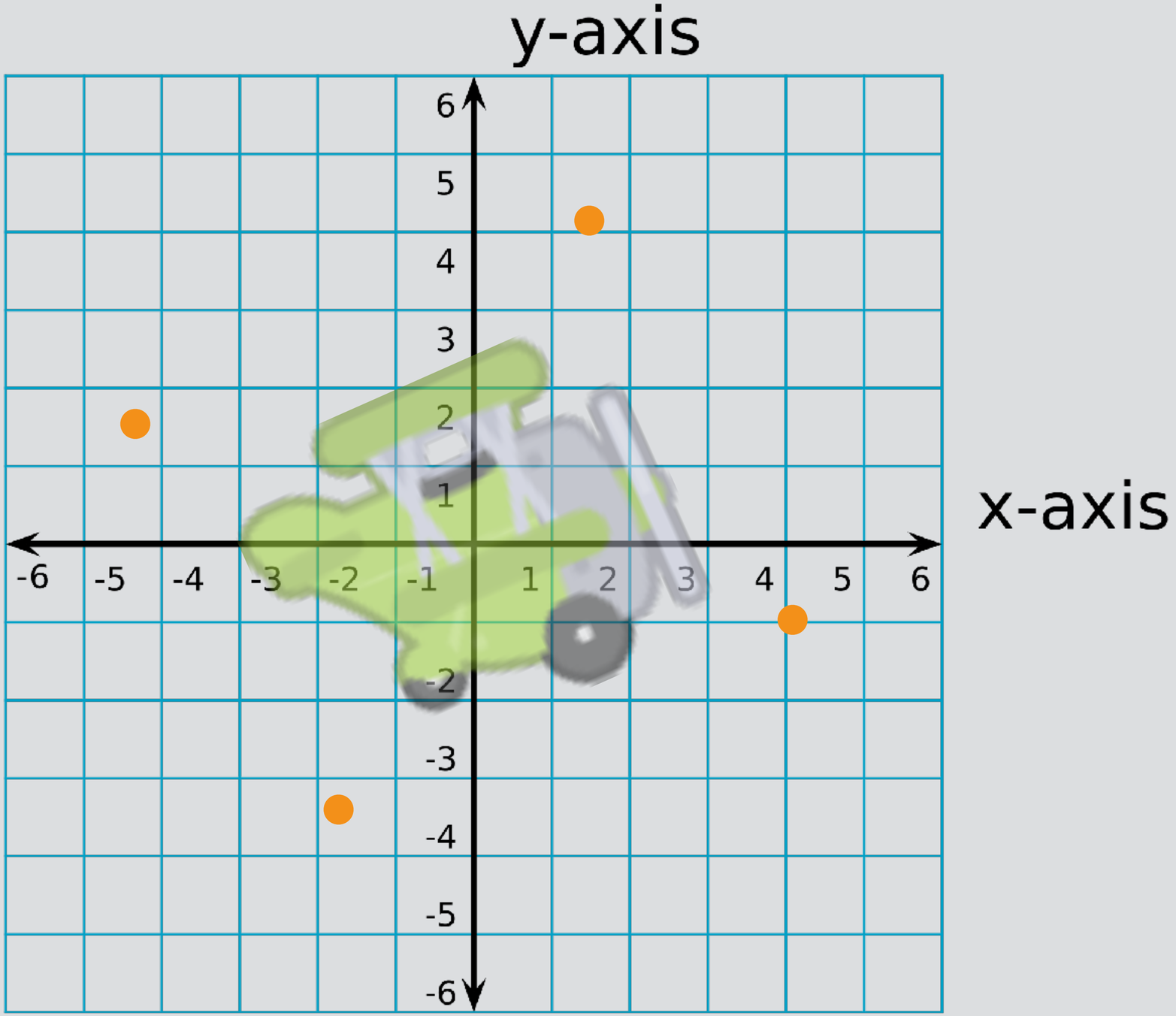
```
void Matrix::Scale (float x, float y, float z);
```

Scales a matrix by specified coordinates.

```
Matrix modelviewMatrix;
modelviewMatrix.Scale(2.0, 2.0, 1.0);
```

Rotation


y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6


y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

```
void Matrix::Rotate(float radians);
```

Rotates a matrix by specified radians.

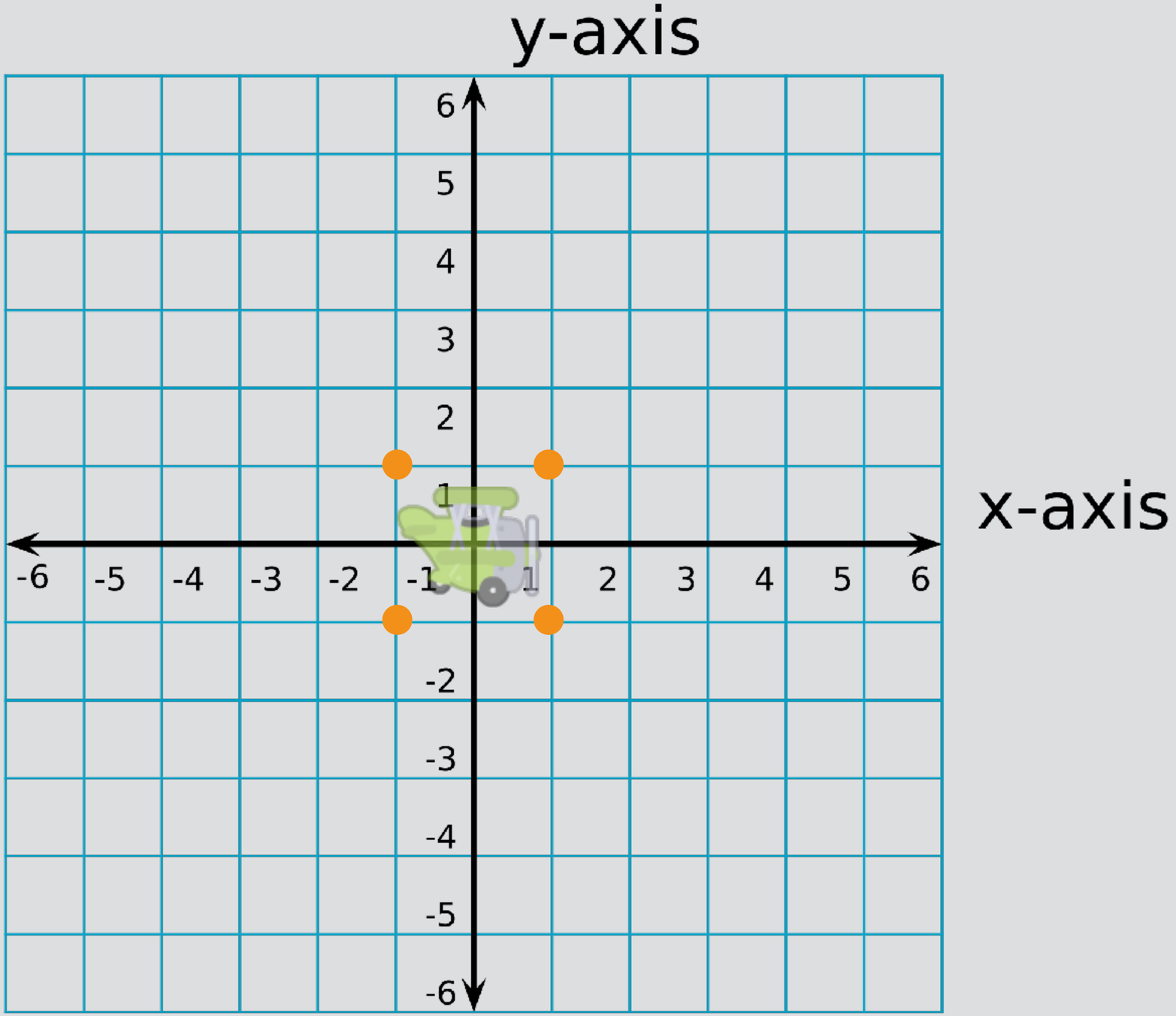
```
Matrix modelviewMatrix;
modelviewMatrix.Rotate(45.0 * (3.1415926 / 180.0));
```

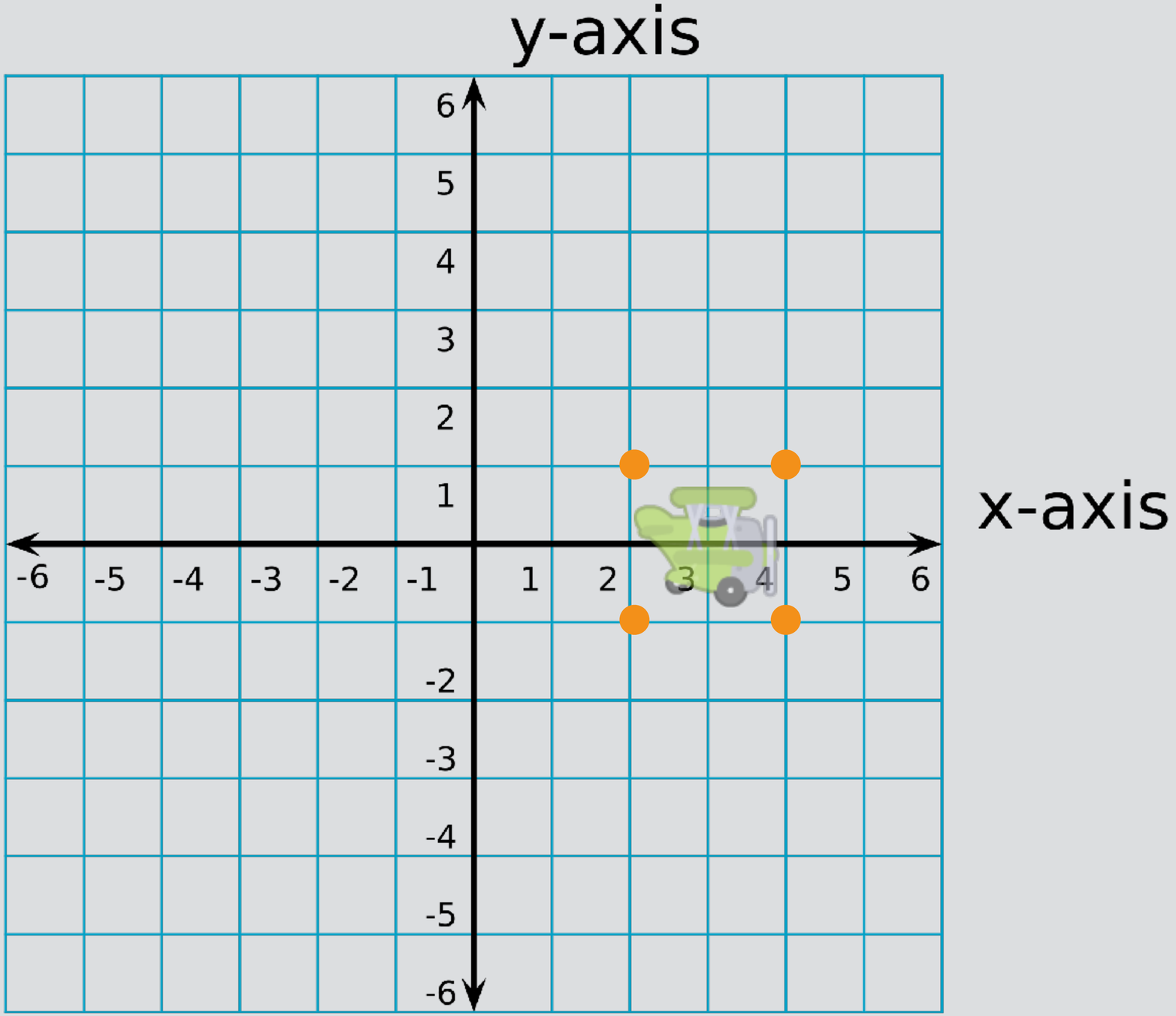
To convert from **degrees** to **radians**:

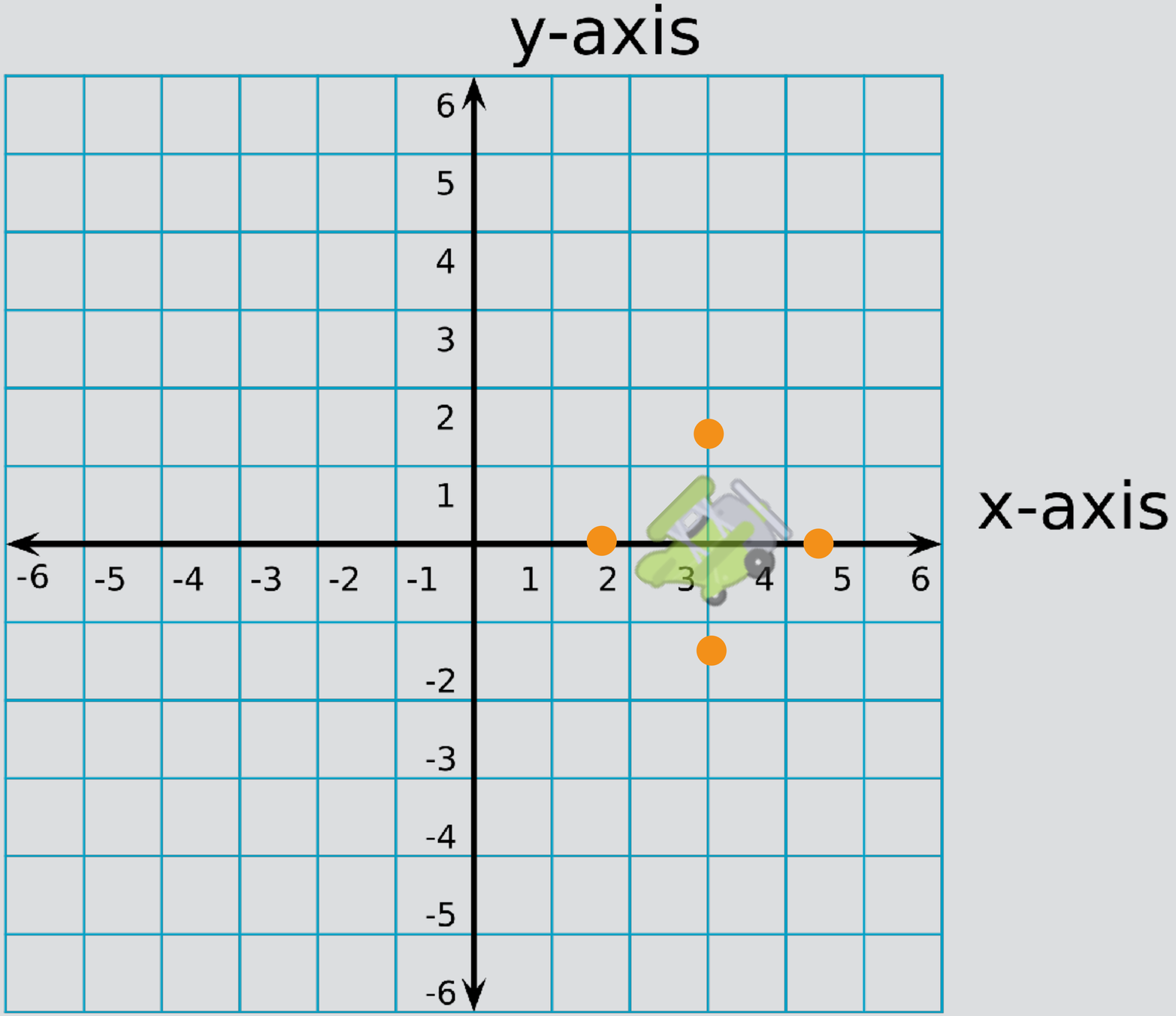
RADIANS = DEGREES * (PI / 180)

Matrix multiplication is not commutative,
so the order of matrix operations matters.

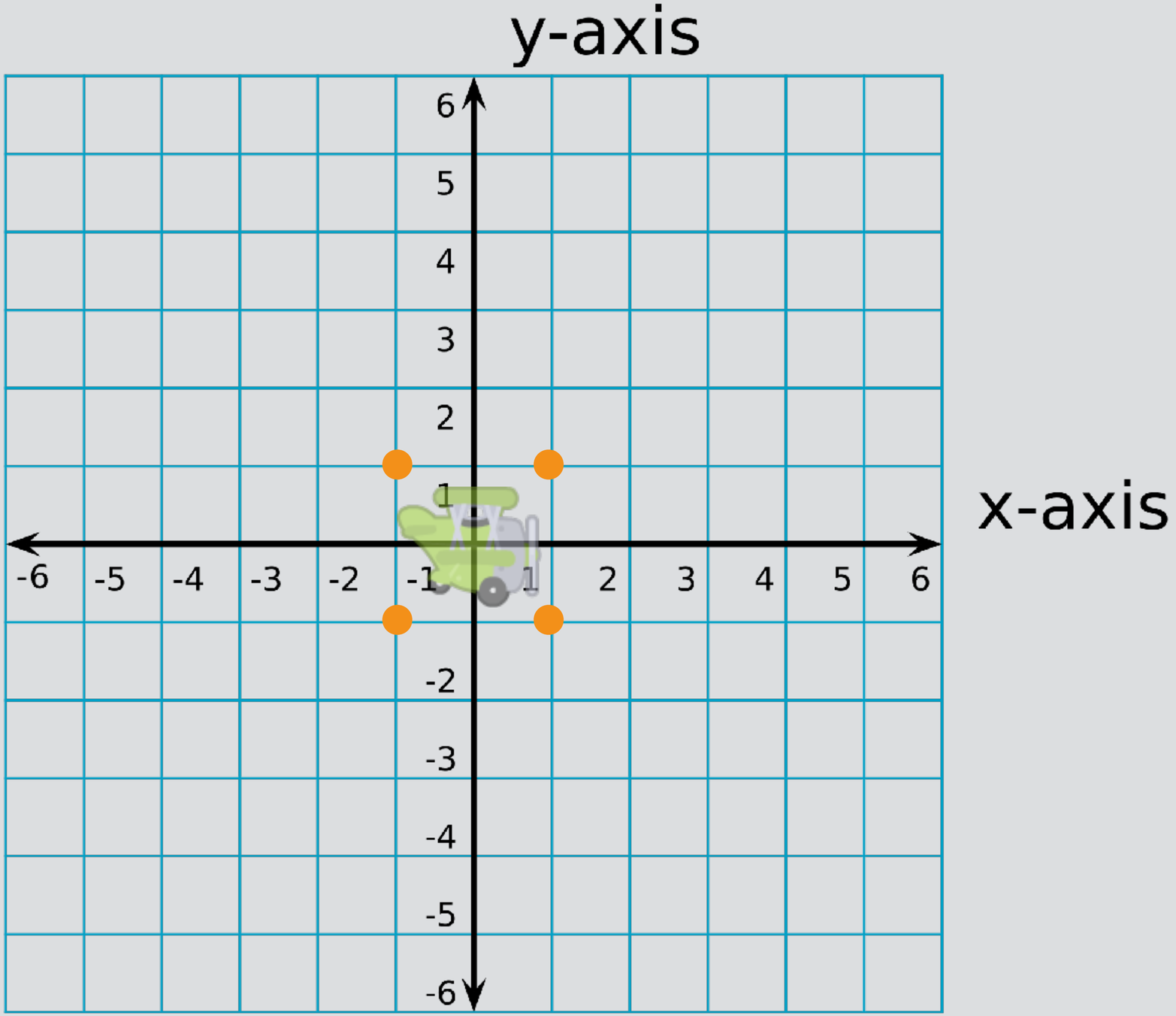
```
Matrix modelviewMatrix;
modelviewMatrix.Translate(2.0f, 0.0f, 0.0f);
modelviewMatrix.Rotate(45.0f * (3.1415926f / 180.0f));
```

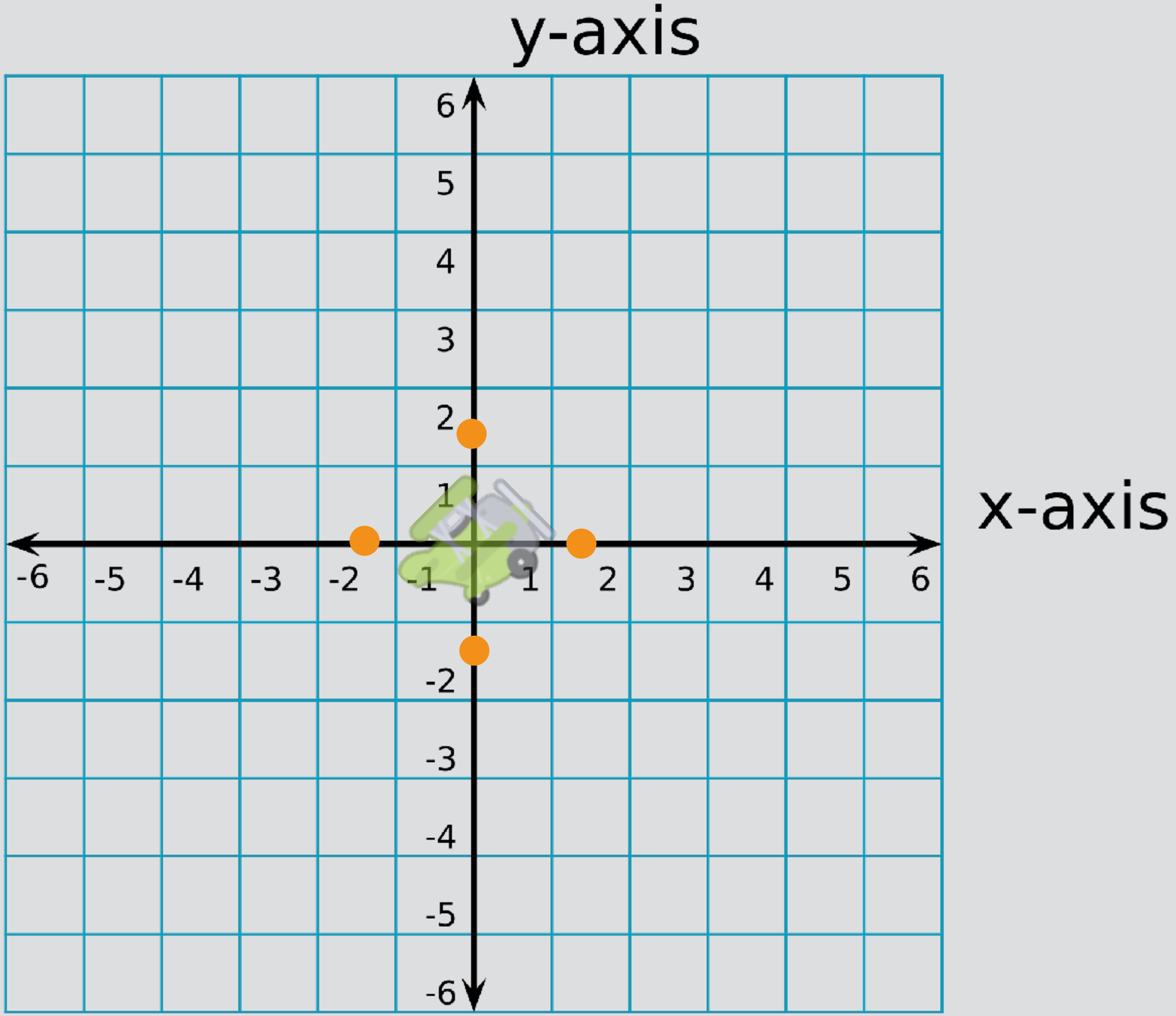

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

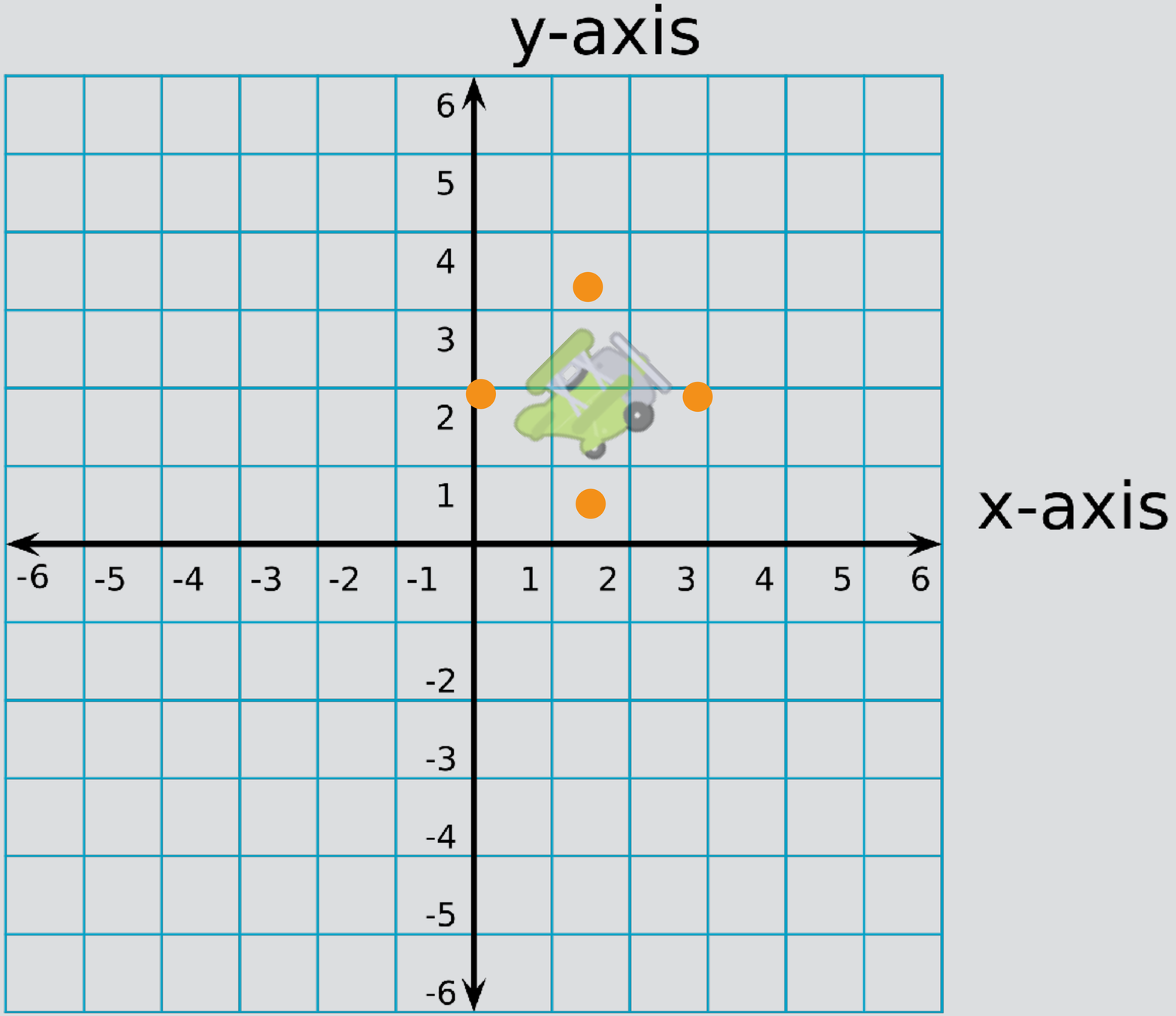

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

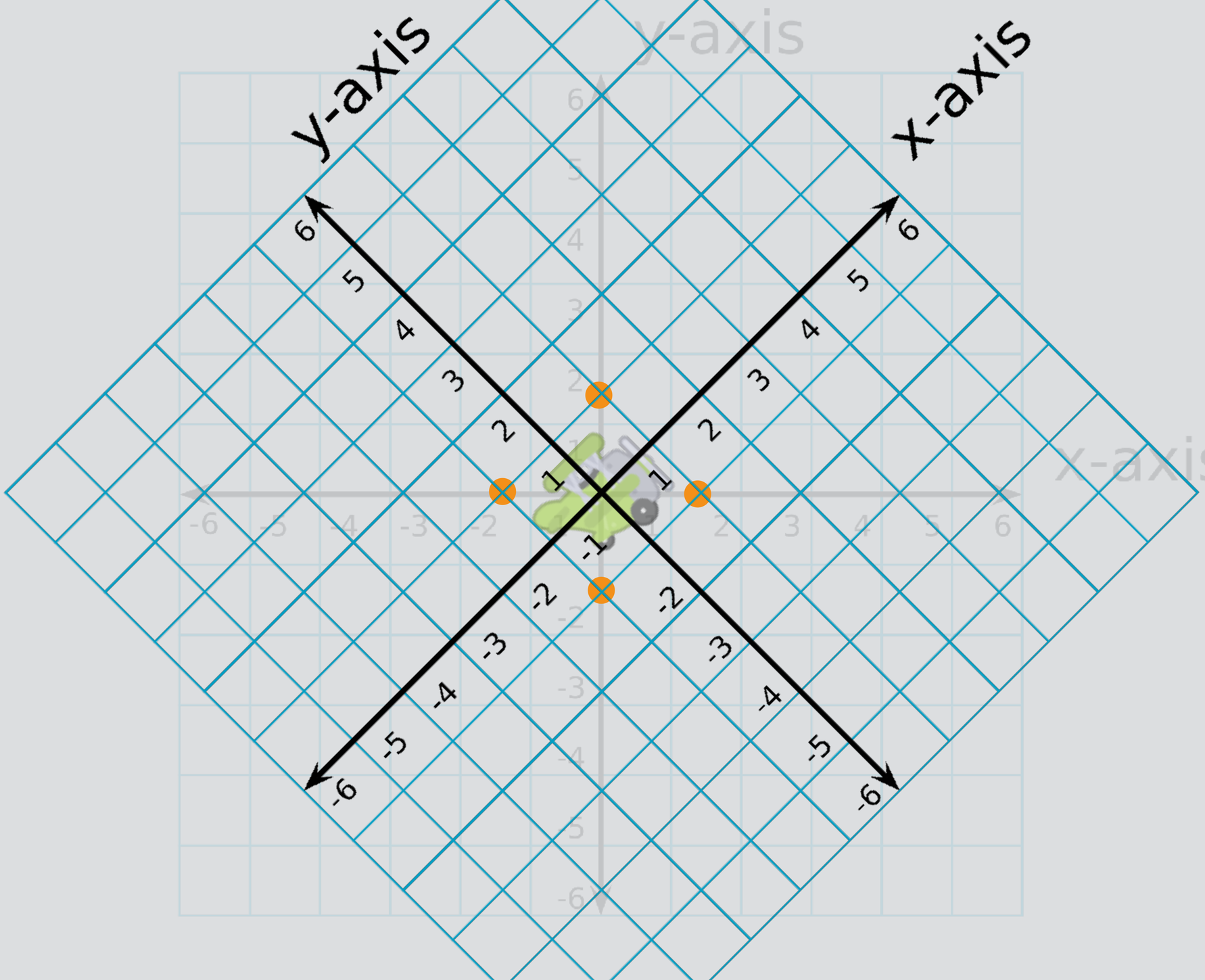

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

```
Matrix modelviewMatrix;
modelviewMatrix.Rotate(45.0f * (3.1415926f / 180.0f));
modelviewMatrix.Translate(2.0f, 0.0f, 0.0f);
```


y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6


y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

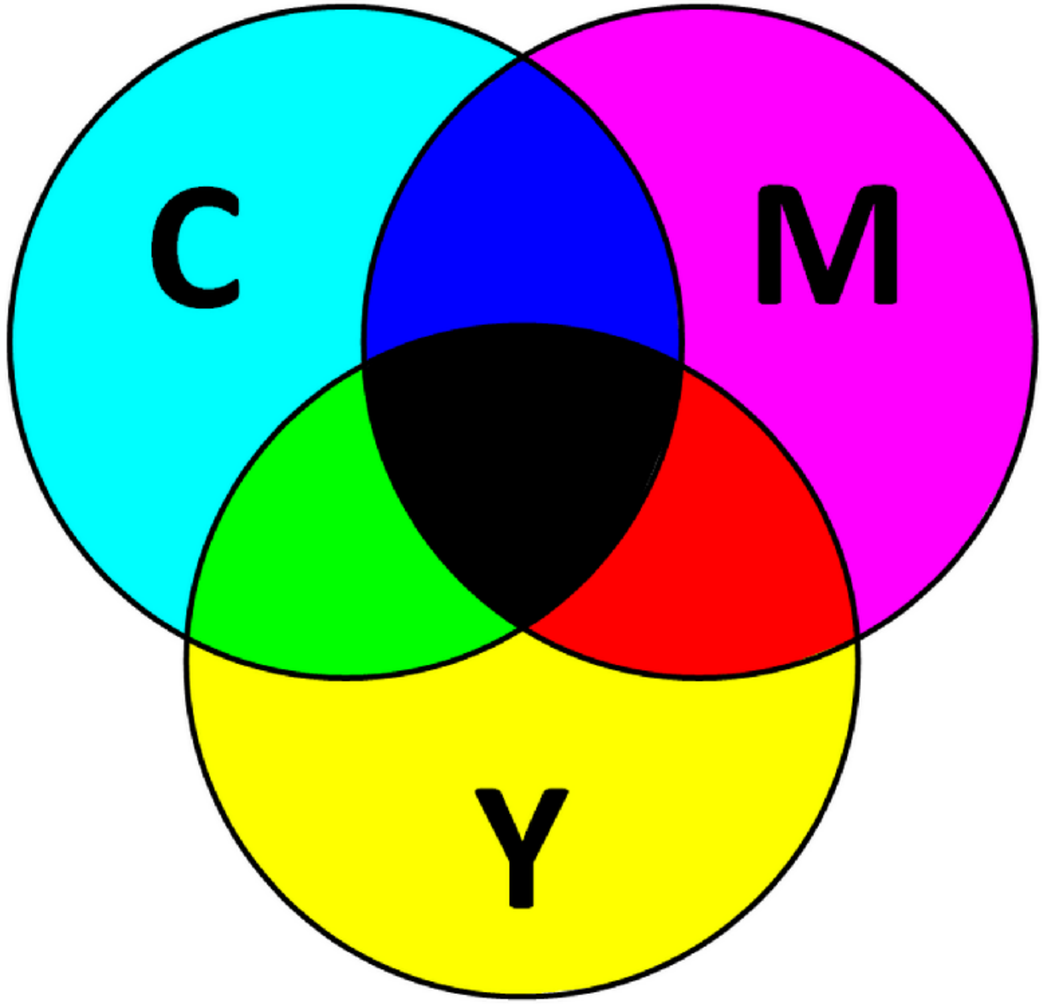

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

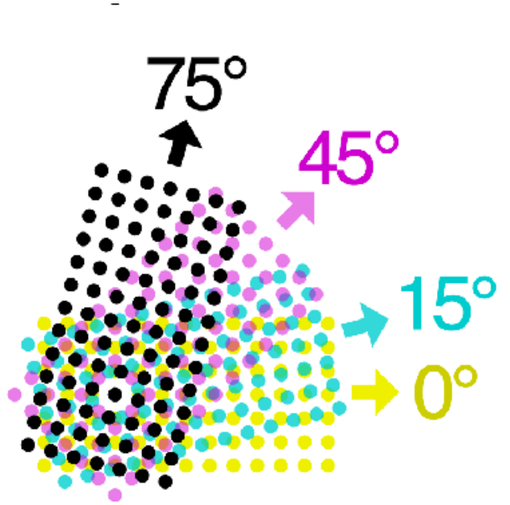

y-axis
x-axis
6
5
4
3
2
1
-1
-2
-3
-4
-5
-6

Color in computer graphics.

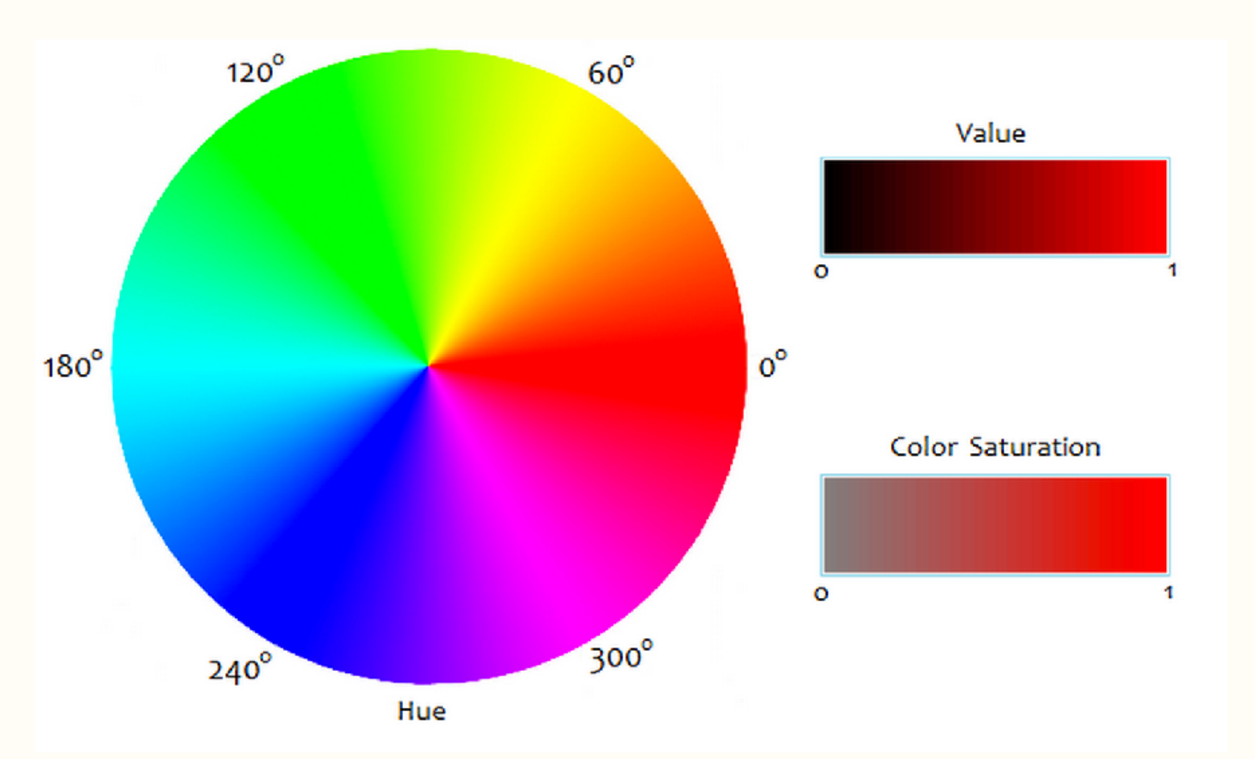
How can color be represented?

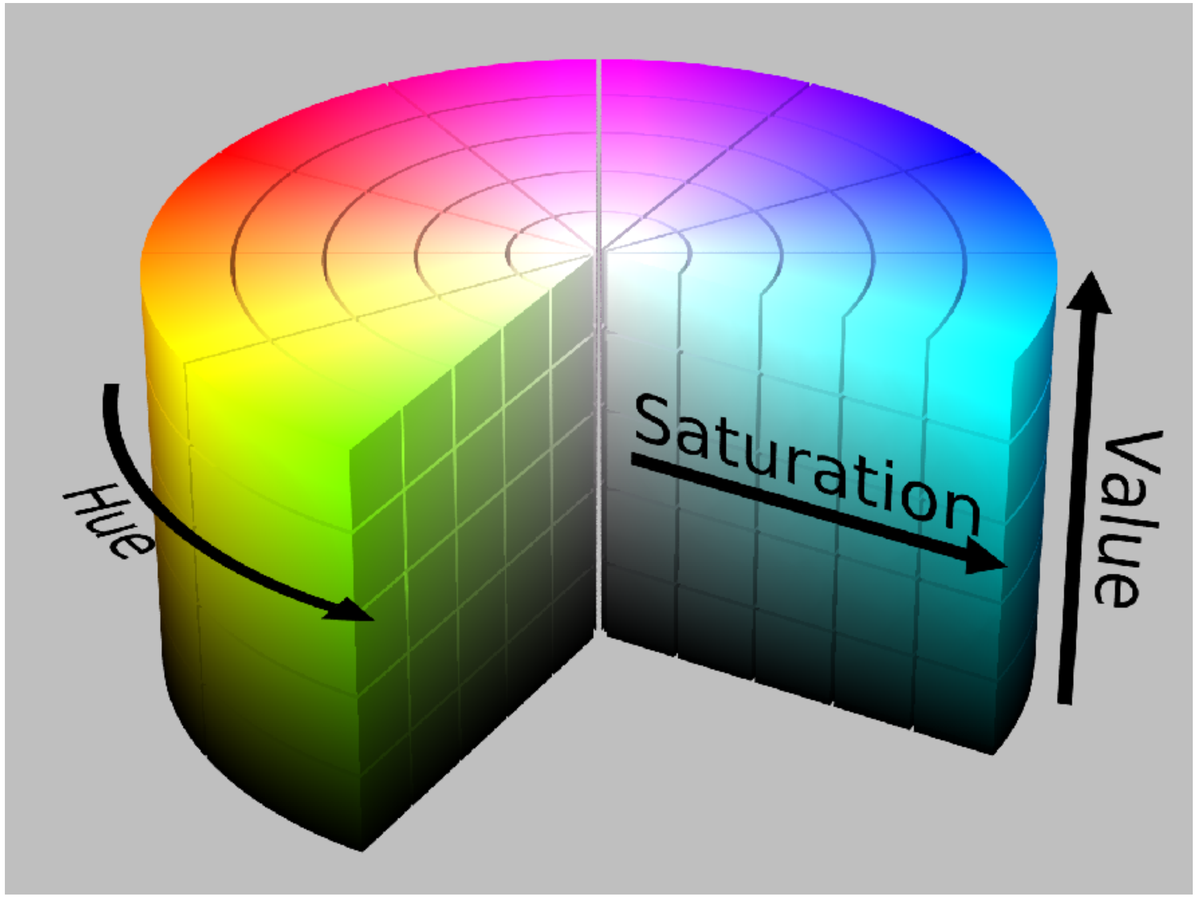
The CMYK Model





The HSV Model





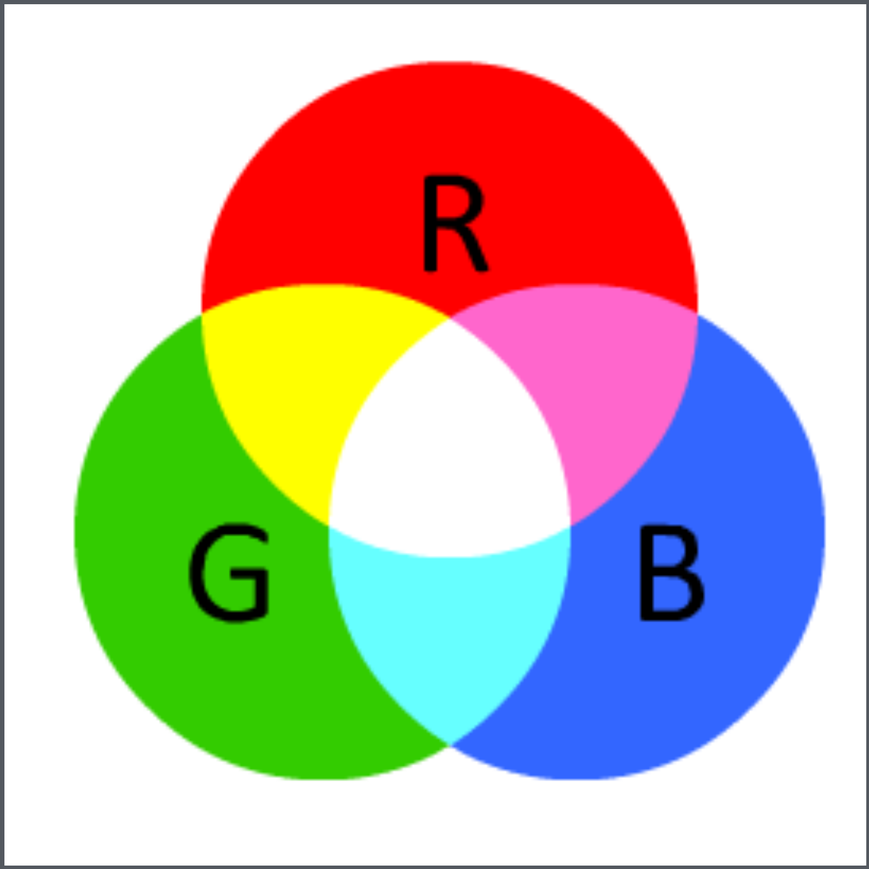
The YUV Model

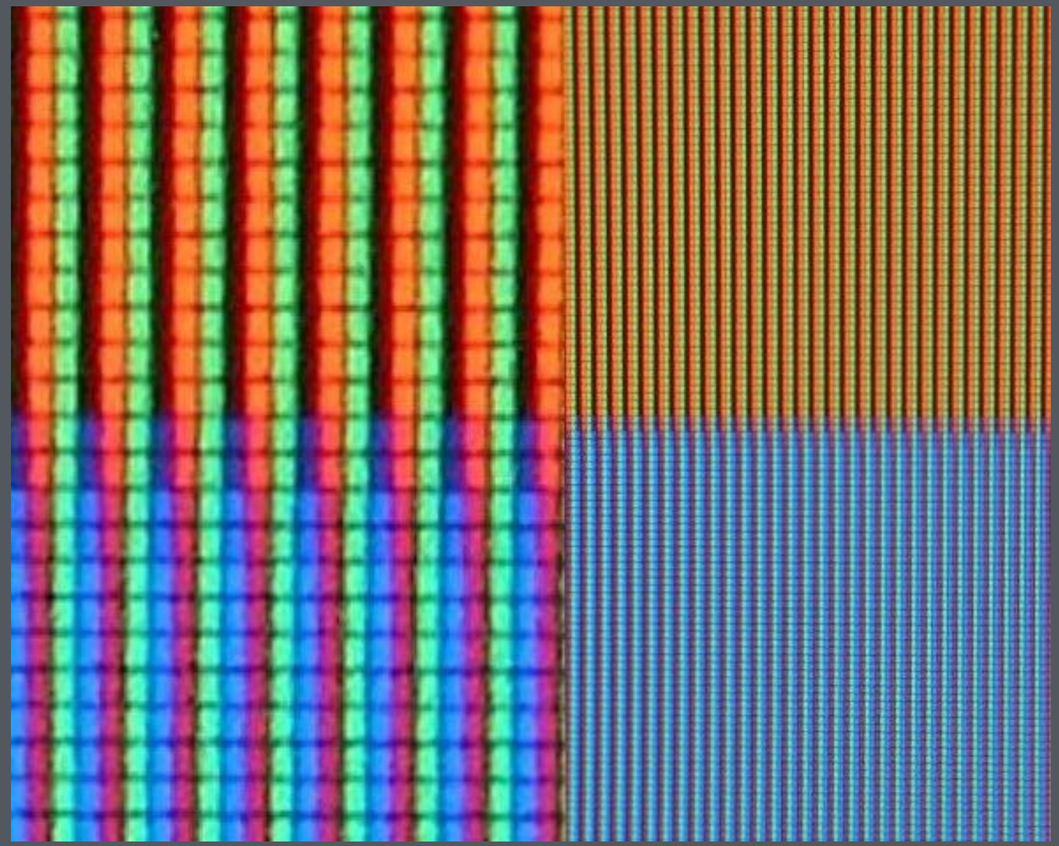
Luminance

Chrominance

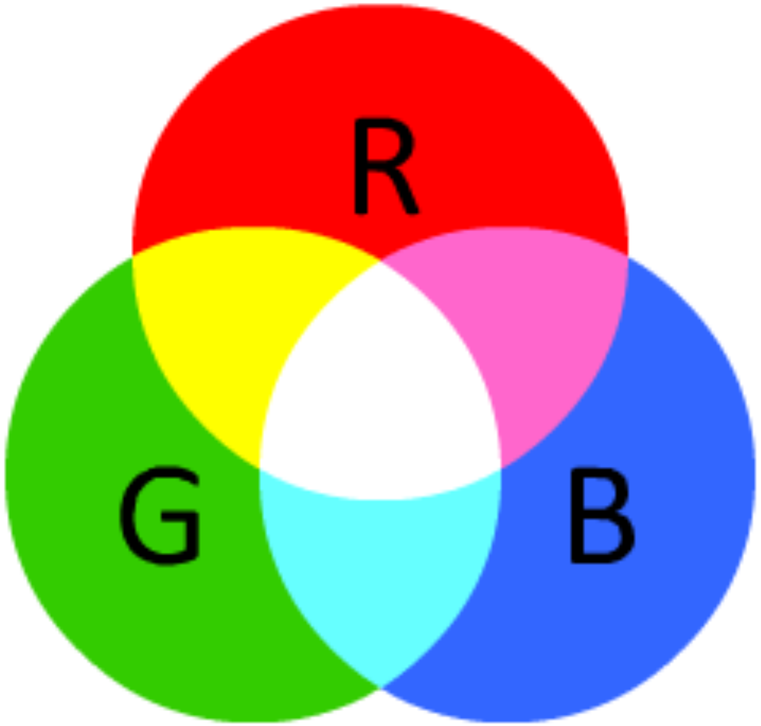
Both

The RGB Model





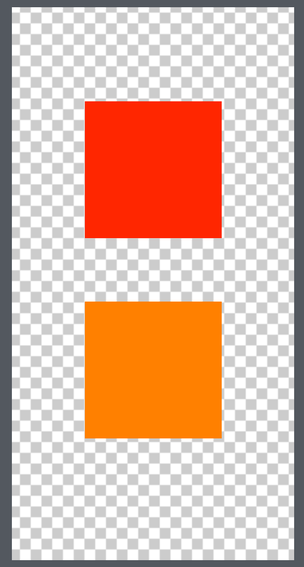
The RGB Model is most common in computer graphics.



Color in OpenGL.

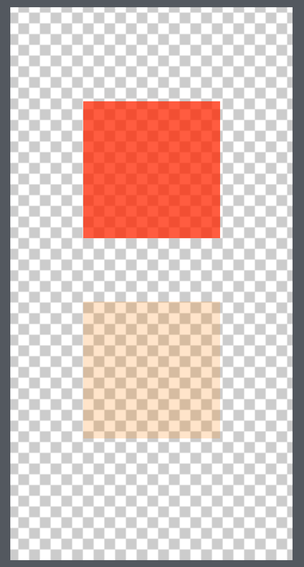
RGB and RGBA colors as 0.0 - 1.0 floating point channels.

RGB



```
1.0,  0.0,  0.0

1.0,  0.5,  0.0
```

RGBA



```
1.0,  0.0,  0.0,  0.7

1.0,  0.5,  0.0,  0.2
```

Clearing the screen.

```
void glClearColor (float red, float green,
float blue, float alpha);
```

Sets the clear color of the screen.

```
glClearColor(0.4f, 0.2f, 0.4f, 1.0f);
```

```
void glClear (GLbitfield mask);
```

Clears the screen to the set clear color.

```
glClear(GL_COLOR_BUFFER_BIT);
```

Textures and images.

Loading an image with STB_image

Include stb_image header.
NOTE: You must define STB_IMAGE_IMPLEMENTATION in one of the files you are including it from!

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

Use stbi_load function to load the pixel data from an image file.

```
int w,h,comp;
unsigned char* image = stbi_load(“some_image.png”, &w, &h, &comp, STBI_rgb_alpha);
```

“w” and “h” will contain the width and height of the image and it will return a pointer to the contents of the image.

After you are done with the image data, you must free it using the stbi_image_free function.

```
stbi_image_free(image);
```

Textures in OpenGL

Creating a texture

```
void glGenTextures (GLsizei numTextures, GLuint *textures);
```

Generates a new OpenGL texture ID.

```
GLuint textureID;
glGenTextures(1, &textureID);
```

Binding a texture

```
void glBindTexture (GLenum target, GLuint texture);
```

Bind a texture to a texture target.

```
glBindTexture(GL_TEXTURE_2D, textureID);
```

Our texture target is always going to be GL_TEXTURE_2D

Setting texture pixel data

```
void glTexImage2D (GLenum target, GLint level, GLint
internalformat, GLsizei width, GLsizei height, GLint
border, GLenum format, GLenum type, const GLvoid *pixels);
```

Sets the **texture data** of the specified **texture target**. Image format must be **GL_RGBA for RGBA images** or **GL_RGB for RGB images**.

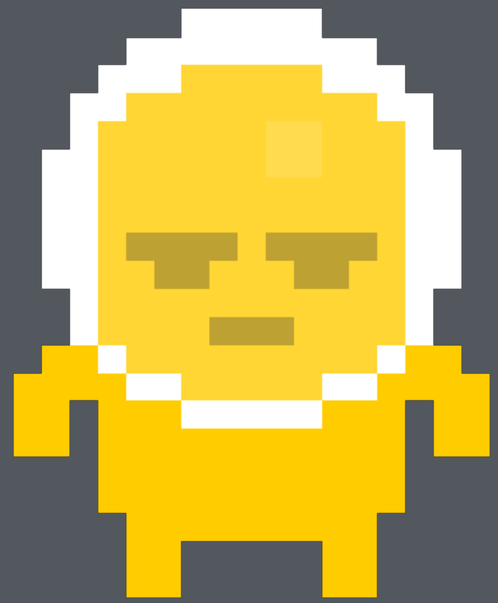
```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
GL_UNSIGNED_BYTE, image);
```

Texture filtering

Texture filtering parameters.

`Linear`

Good for high resolution textures.



`Nearest neighbor`

Good for pixelart.

```
void glTexParameteri (GLenum target, GLenum pname,
GLint param);
```

Sets a texture parameter of the specified texture target.
We MUST set the texture filtering parameters GL_TEXTURE_MIN_FILTER and GL_TEXTURE_MAG_FILTER before the texture can be used.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

Use GL_LINEAR for linear filtering and GL_NEAREST for nearest neighbor filtering.

Putting it all together.

```
GLuint LoadTexture(const char *filePath) {
    int w,h,comp;
    unsigned char* image = stbi_load(filePath, &w, &h, &comp, STBI_rgb_alpha);

    if(image == NULL) {
        std::cout << "Unable to load image. Make sure the path is correct\n";
        assert(false);
    }

    GLuint retTexture;
    glGenTextures(1, &retTexture);
    glBindTexture(GL_TEXTURE_2D, retTexture);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    stbi_image_free(image);
    return retTexture;
}
```

Texture coordinates.

Texture coordinates are defined in 0-1 units called UV coordinates, not pixels!

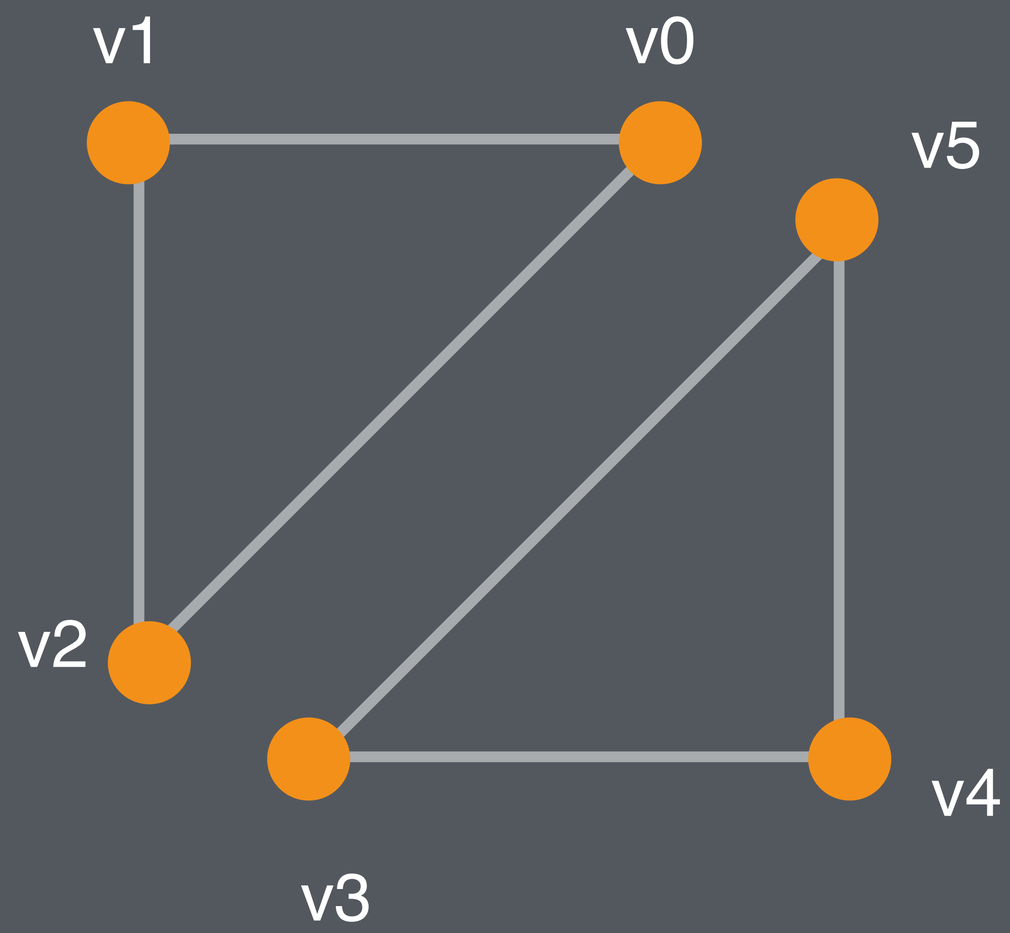
Defining UV coordinate attributes.

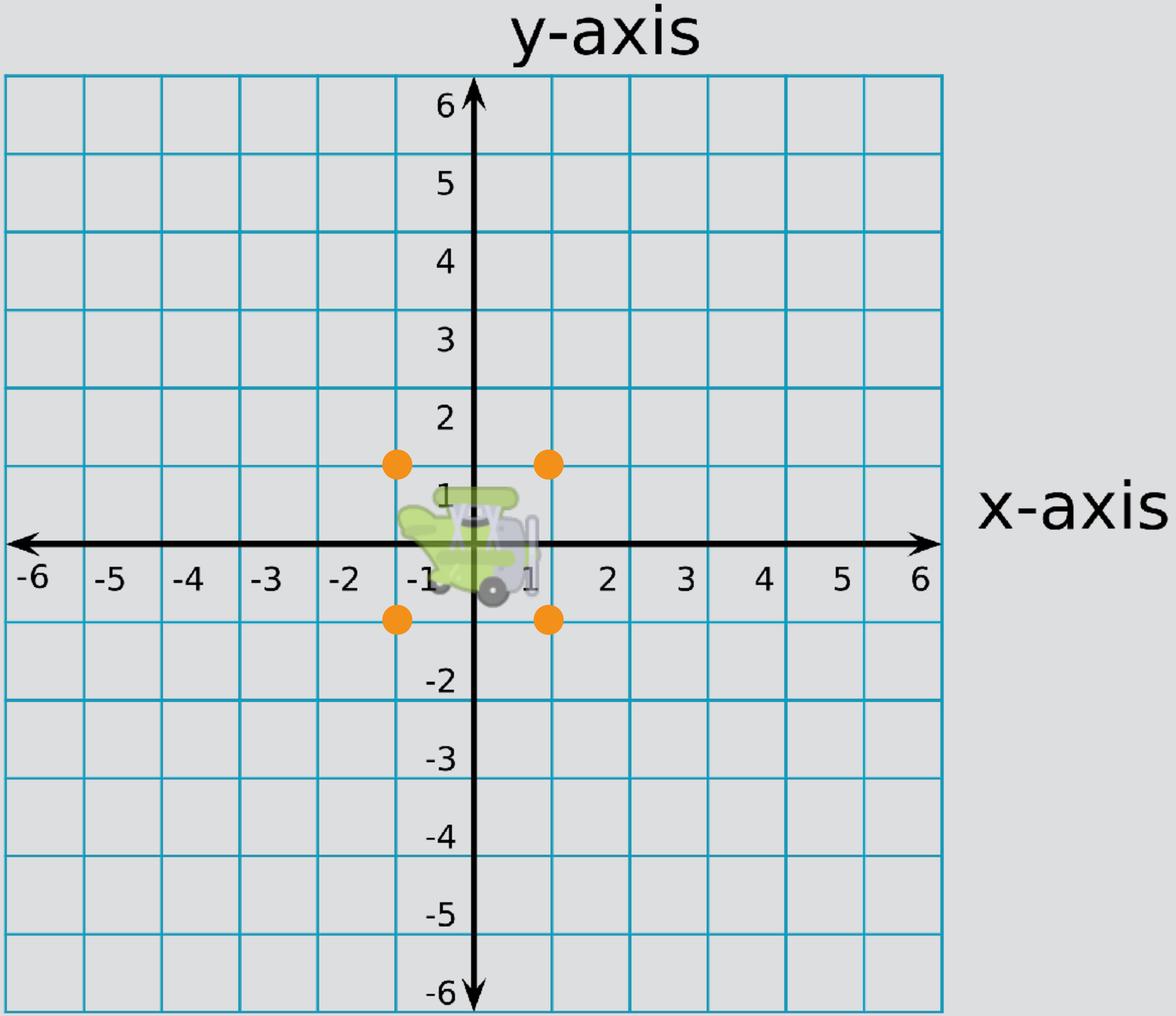
```
void glVertexAttribPointer (GLint index, GLint
size, GLenum type, GLboolean normalized, GLsizei
stride, const GLvoid *pointer);
```

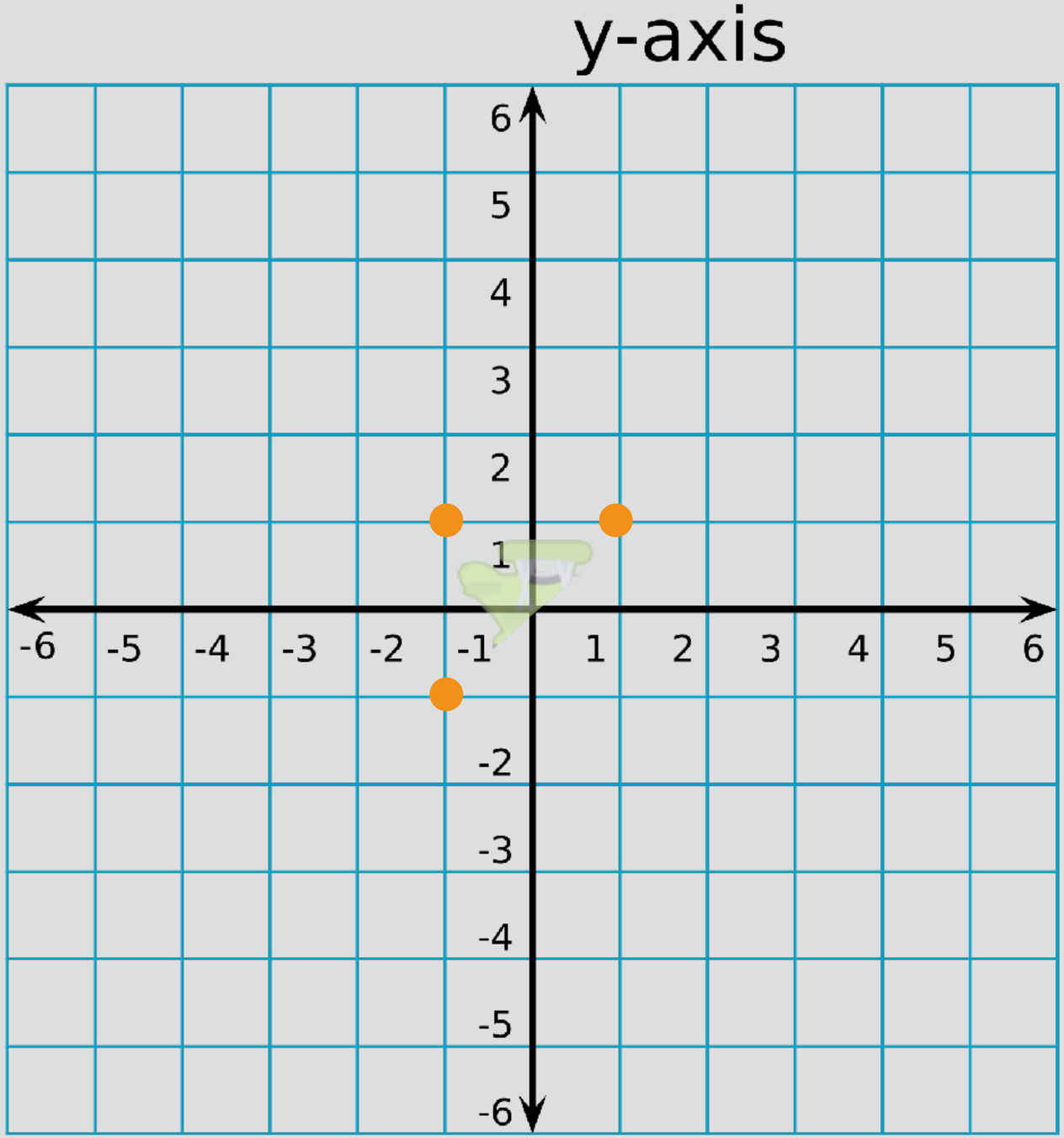
Defines an array of **vertex data**.

```
float texCoords[] = {0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f,};

glVertexAttribPointer(program.texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords);
```

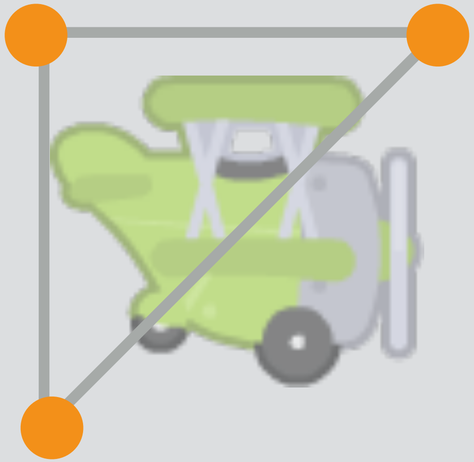
Drawing a sprite.

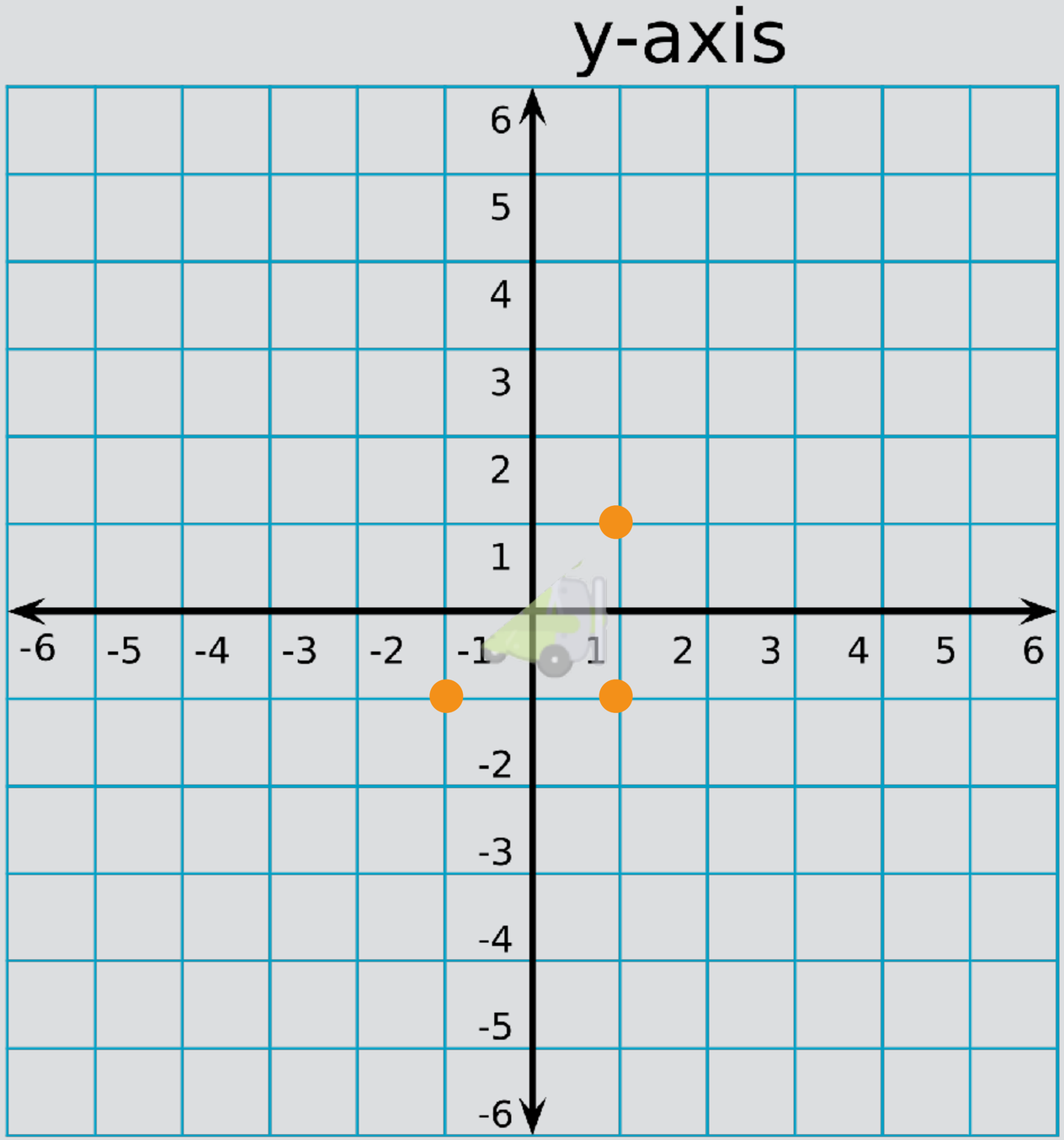

v1
v0
v5
v2
v4
v3


y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

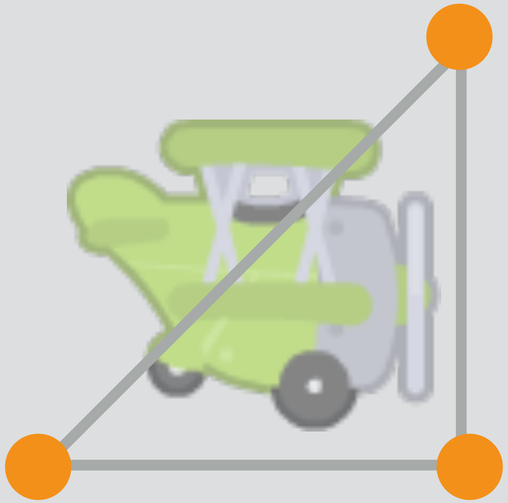


x-axis





x-axis



Drawing a sprite.

- Set position attributes for 2 triangles.
- Set texture coordinate attributes for 2 triangles.
- Bind the texture we want to use.
- Draw arrays.
- Disable attribute arrays.

Need to use a shader program that supports textures!

```
ShaderProgram program(RESOURCE_FOLDER”vertex_textured.glsl”, RESOURCE_FOLDER”fragment_textured.glsl");
```

Putting it all together.

Setup (before the loop)

```
glViewport(0, 0, 640, 360);

ShaderProgram program(RESOURCE_FOLDER"vertex_textured.glsl", RESOURCE_FOLDER"fragment_textured.glsl");

GLuint emojiTexture = LoadTexture(RESOURCE_FOLDER"emoji.png");

Matrix projectionMatrix;
projectionMatrix.SetOrthoProjection(-3.55f, 3.55f, -2.0f, 2.0f, -1.0f, 1.0f);
Matrix modelviewMatrix;
```

Drawing (in your game loop)

```
glUseProgram(program.programID);

program.SetModelviewMatrix(modelviewMatrix);
program.SetProjectionMatrix(projectionMatrix);

glBindTexture(GL_TEXTURE_2D, emojiTexture);

float vertices[] = {-0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5};
glVertexAttribPointer(program.positionAttribute, 2, GL_FLOAT, false, 0, vertices);
glEnableVertexAttribArray(program.positionAttribute);

float texCoords[] = {0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0};
glVertexAttribPointer(program.texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords);
glEnableVertexAttribArray(program.texCoordAttribute);

glDrawArrays(GL_TRIANGLES, 0, 6);

glDisableVertexAttribArray(program.positionAttribute);
glDisableVertexAttribArray(program.texCoordAttribute);

SDL_GL_SwapWindow(displayWindow);
```

Blending

Blending

Enabling blending

```
glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Keeping time.

In **setup**

```
float lastFrameTicks = 0.0f;
```

In **game loop**

```
float ticks = (float)SDL_GetTicks()/1000.0f;
float elapsed = ticks - lastFrameTicks;
lastFrameTicks = ticks;
```

elapsed is how many seconds **elapsed since last frame**.

We will use this value to **move everything** in our game.

Basic time-based animation.

```
angle += elapsed;

// rotate matrix by angle
// draw sprite
```

Assignment #1

- Create a simple 2D scene using textured polygons.
- You can use any images you want, but feel free to use the assets in the class github repo.
- At least one element must be animated (rotating, scaling or translating based on time).
- You must use at least 3 different textures.
- Commit the source to your github repository and email me the link.