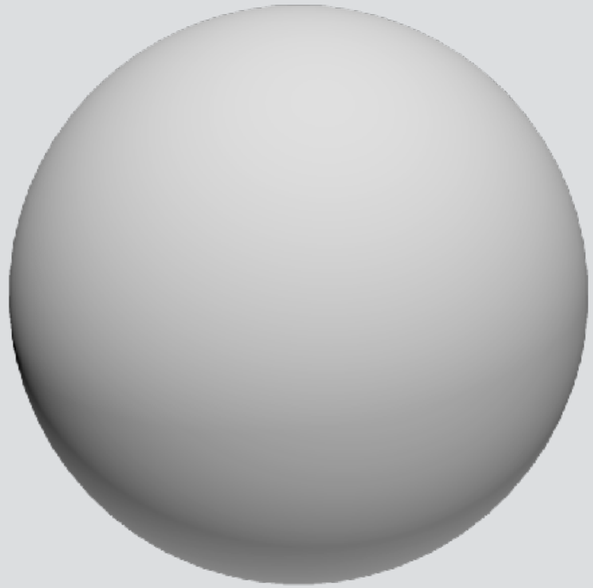
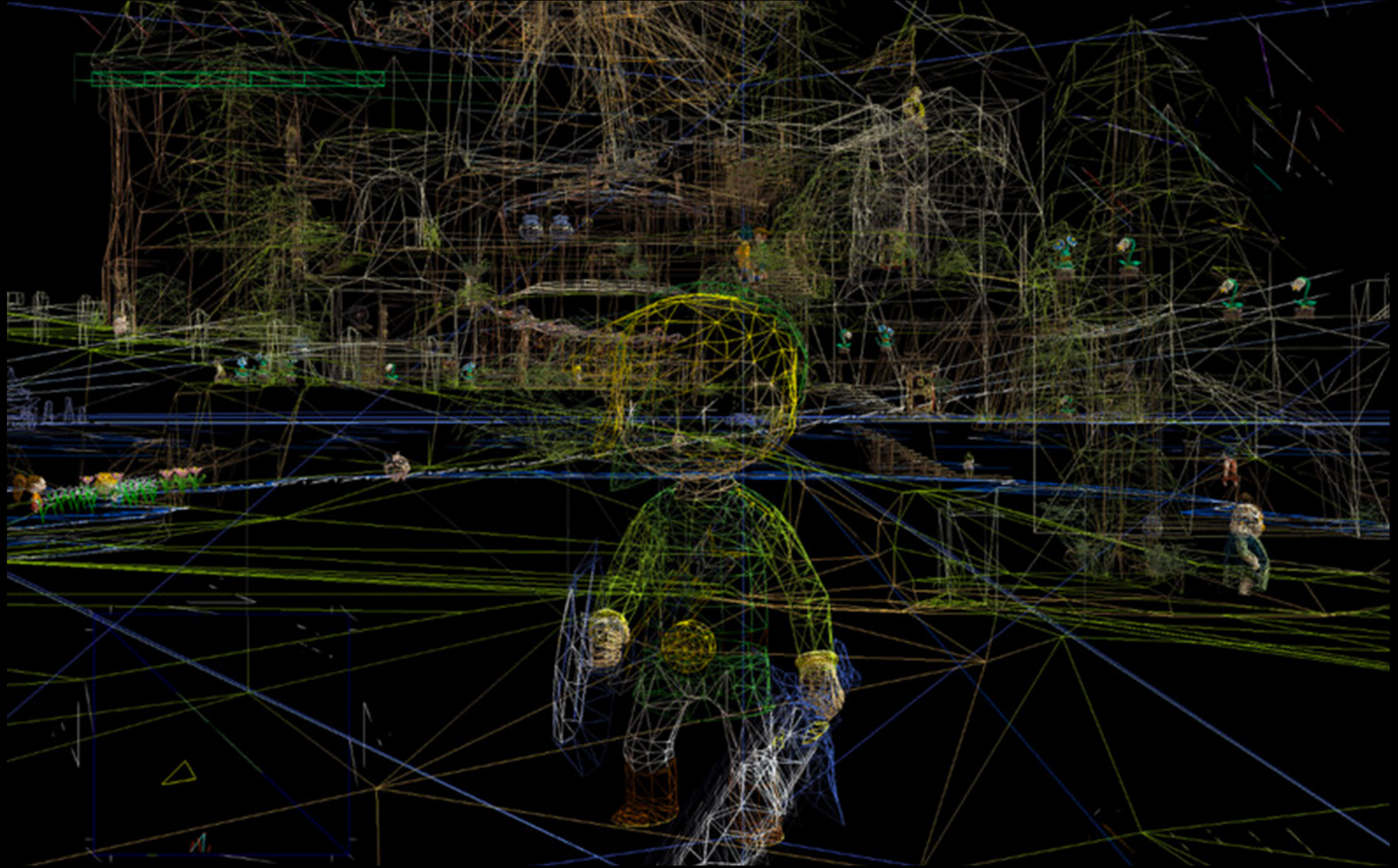
Graphics Foundations Part 1

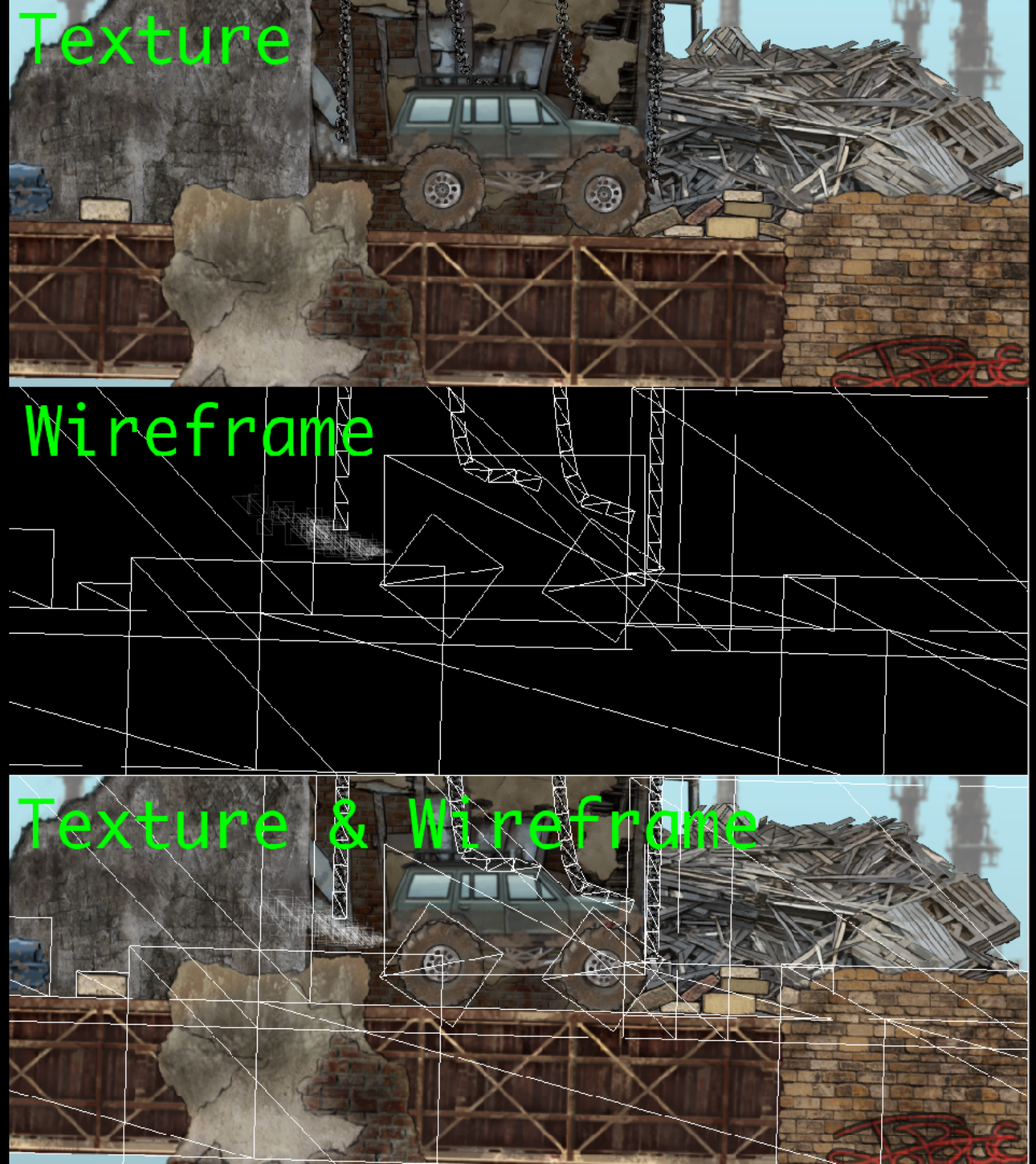


CS 3113

Computer graphics


Defend
Put Away
72
FREE
4168




Texture
Wireframe
Texture & Wireframe

Digital images

Image on the screen

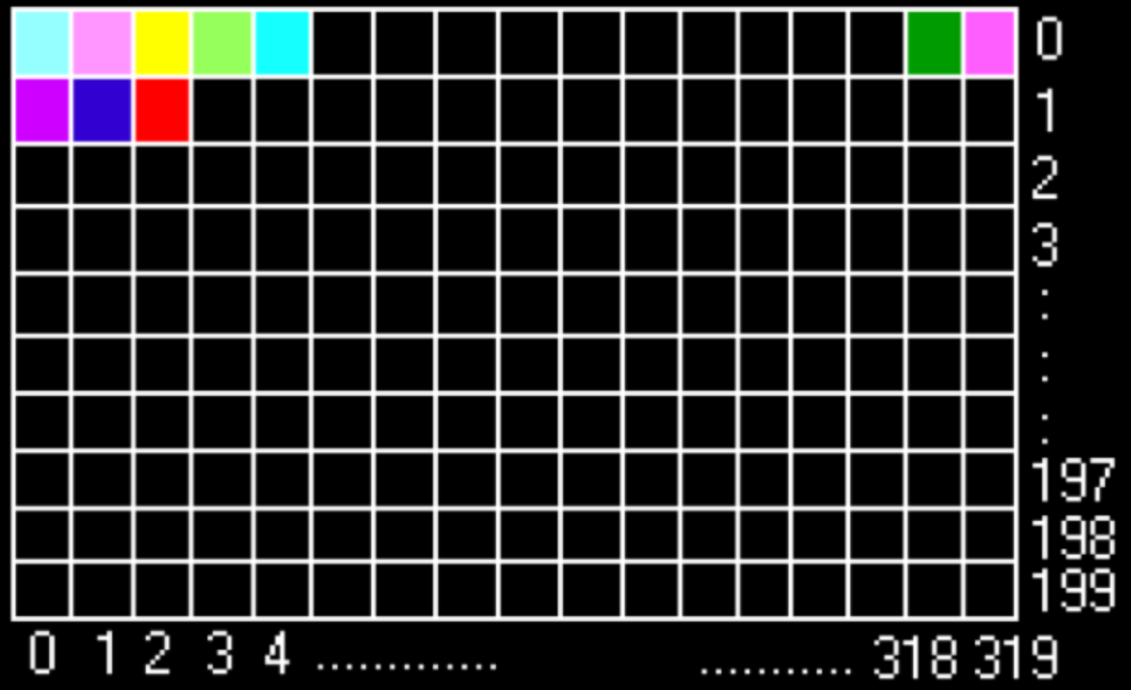
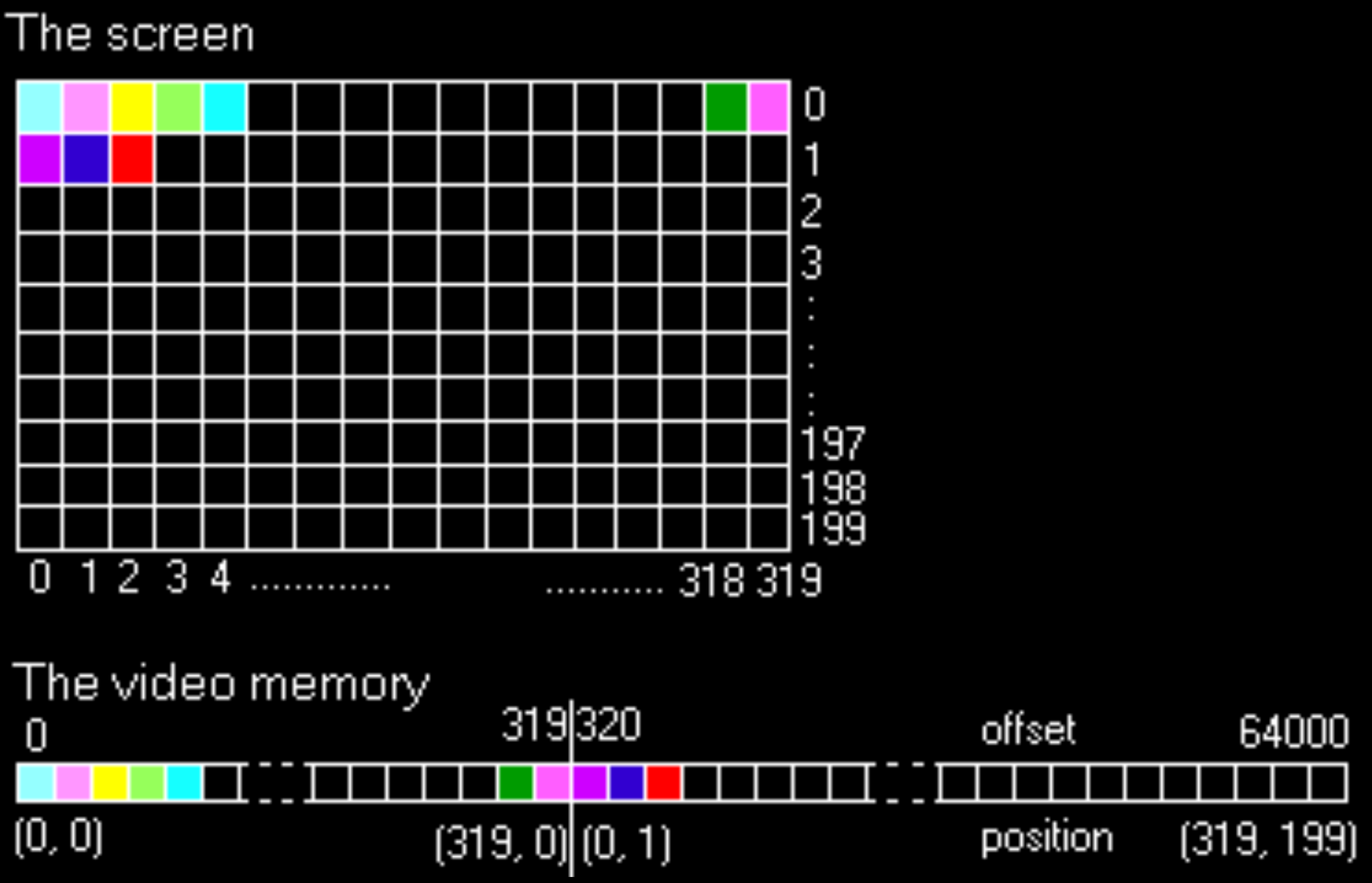


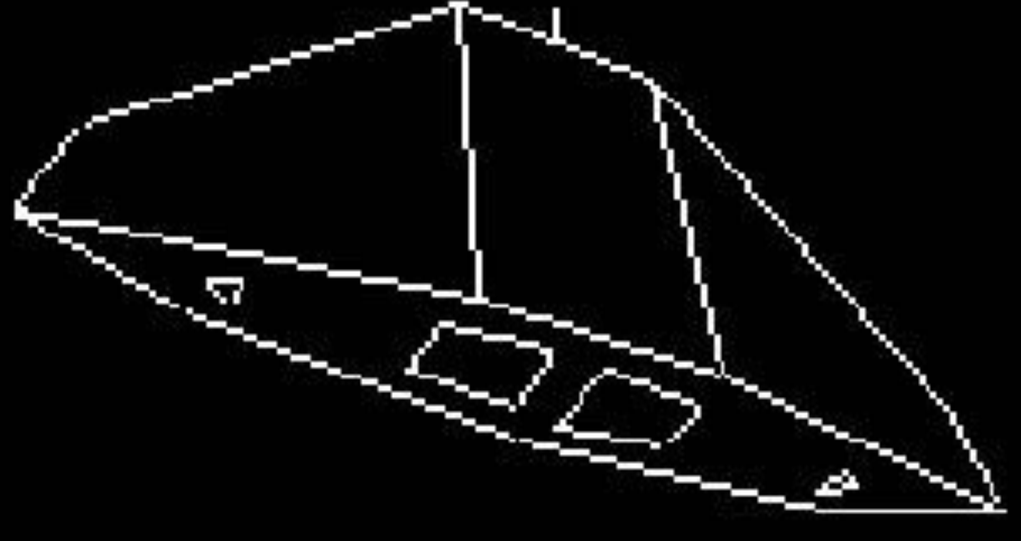
Image in memory

Graphics in games

Software rendering


The screen
0
1
2
3
197
198
199
0 1 2 3 4 .... 318 319
The video memory
0
319
320
offset
64000
(0, 0)
(319, 0)
(0, 1)
position
(319, 199)

Software rendering



Load New Commander (Y/N)?

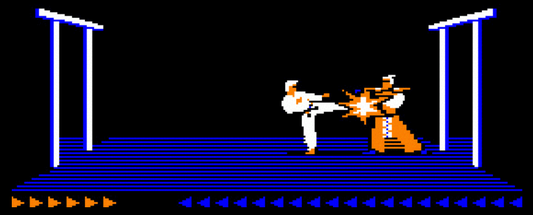

SPD: 0
VSI: 01.78
THR:90%
HDG:0
ALT:83

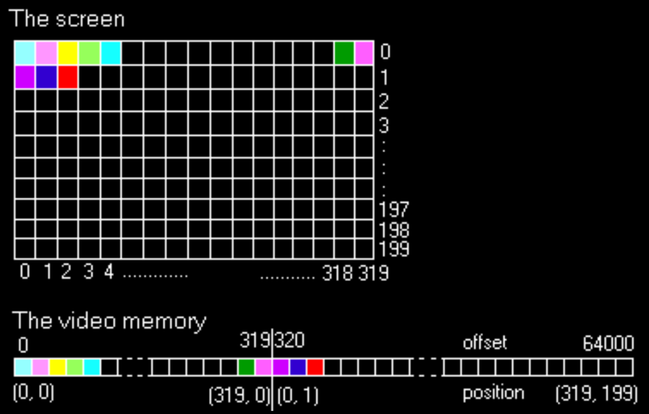

93
75
13
0
78
91
13


LEVEL UP
CHAR
QUESTS
MAP
MENU
1
2
3
4
5
6
7
8
RED STORM
TOTAL KILLS : 19
RESISTS : LIGHTNING
IMMUNE : MAGIC
INV
SPELLS

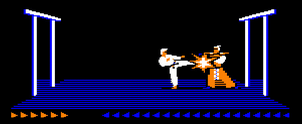
Software Rendering

RAM

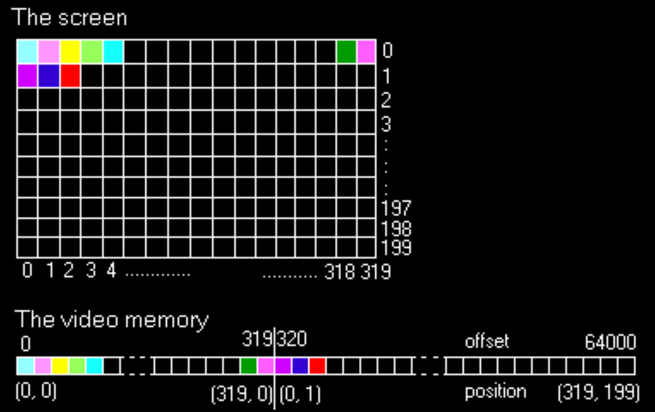




Video card



Software Rendering



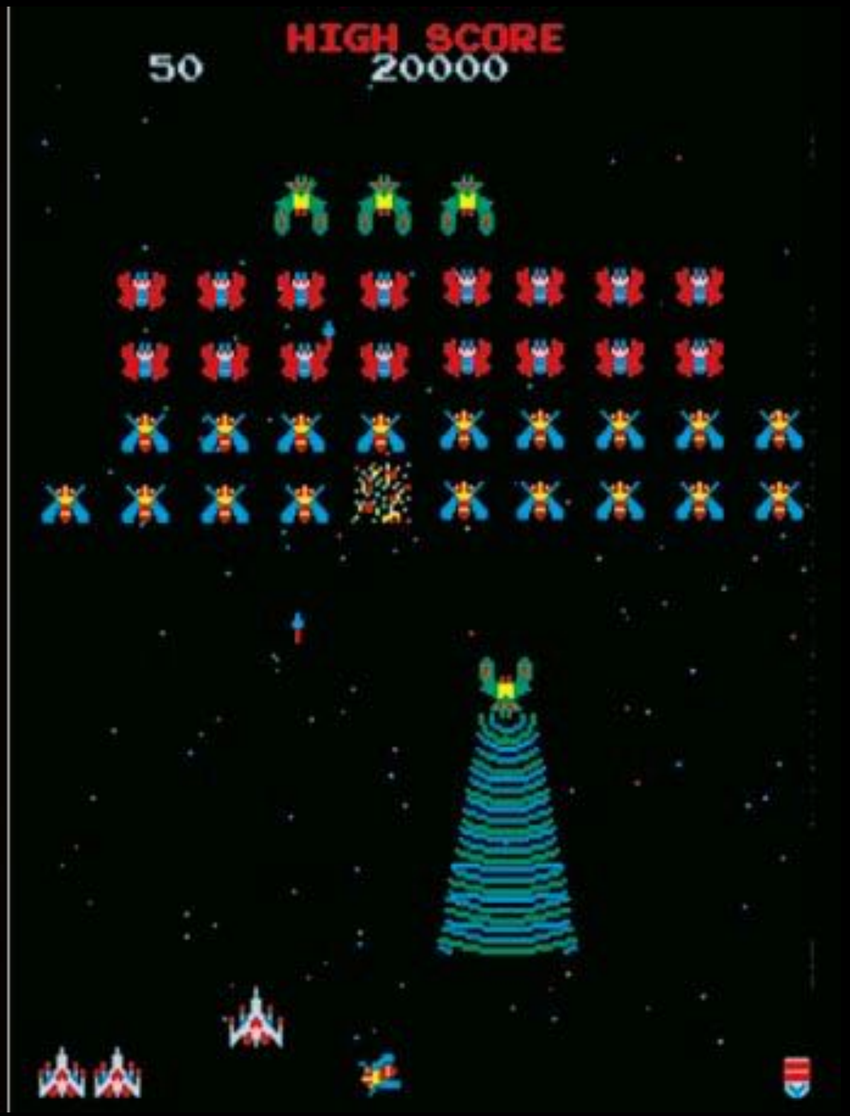
2560 * 1600 = 4,096,000 pixels

Hardware rendering


Galaga
DOUBLE DONKEY KONG
Nintendo
MS. PAC-MAN
MIDWAY
Bally
Home Theater of Arcades


HIGH SCORE
1050
1050


1UP
340
HIGH SCORE
340


HIGH SCORE
50
20000


Nintendo
ENTERTAINMENT SYSTEM™
POWER
RESET
Nintendo

MARIO
000000

×00

WORLD
1-1

TIME

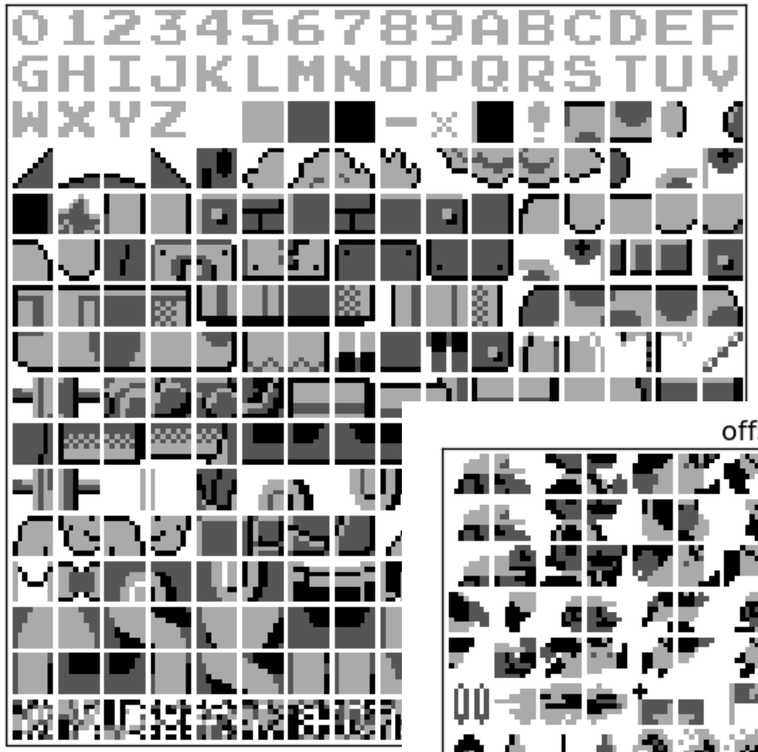
SUPER MARIO BROS.

1 PLAYER GAME

2 PLAYER GAME

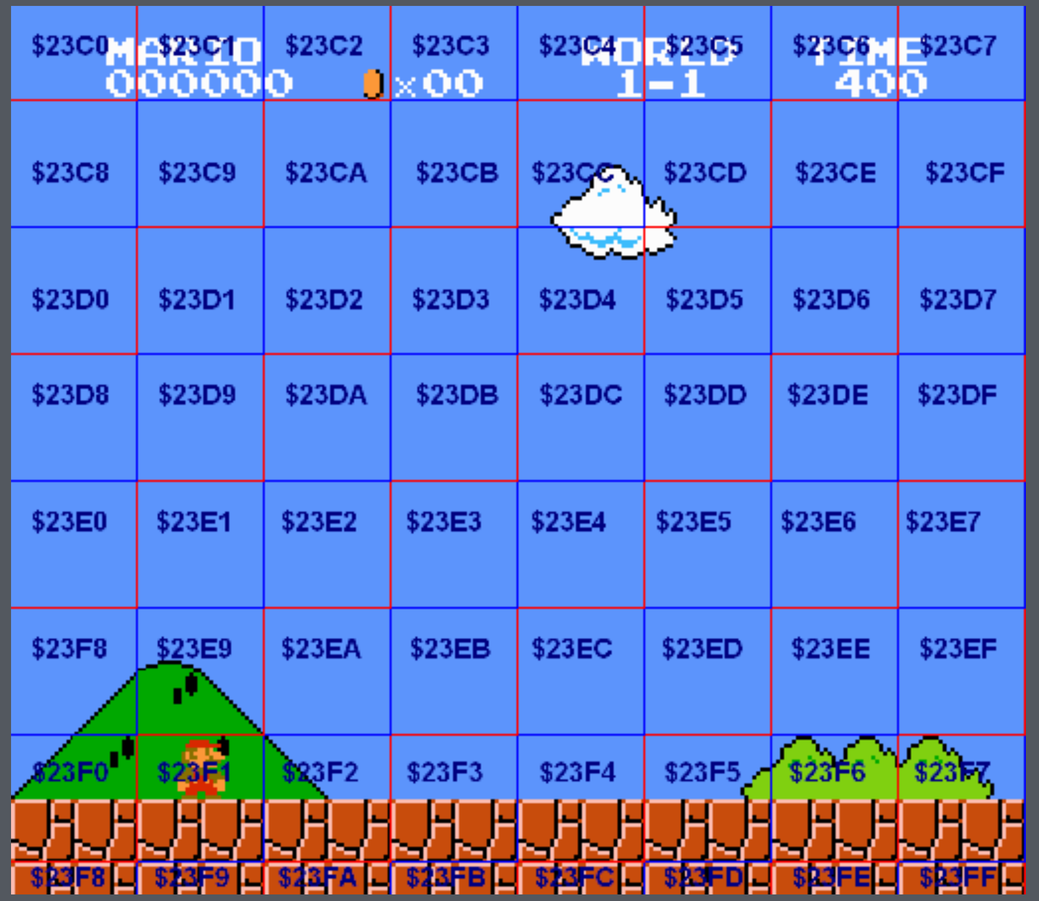
TOP- 000000

offset = 2305

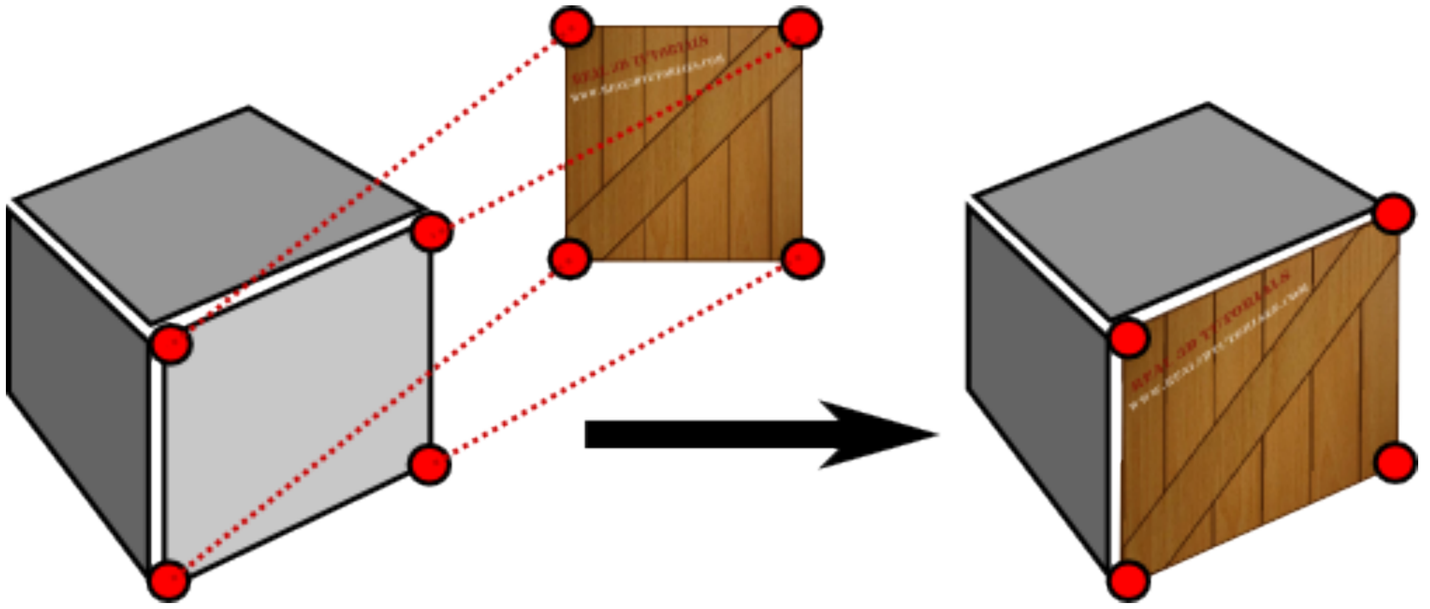


offset = 2049

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F




DAYTONA
USA
RACE LEADER
RACE LEA
50
2ND/2
50
1ST/2




12MB
BLASTER
Hardware XTRA AWARD
PC-Games 05/98
Hardware XTRA AWARD
PC-Action 05/98
PC Player
PC-Player 05/98
FASZINIERENDE SPIELE
4
FASZINIERENDE SPIELE
Voodoo 2
Get the Magic of Speed!
Basiert auf dem neuen Voodoo2 Graphics™ Chipsatz von 3Dfx Interactive™
Ausgestattet mit 12 MB Hochleistungsspeicher für stärkste Leistung
Bis zu 50 Milliarden Operationen und 3 Millionen Triangles pro Sekunde
Bis zu dreimal schnellere 3D-Verarbeitung als beim ursprünglichen Voodoo Graphics™-Chipsatz!
Arbeitet mit Ihrer vorhandenen Grafikkarte zusammen und bietet Ihnen das schnellste 3D-Spiel aller Zeiten
Enthält 4 topaktuelle Spiele
INCOMING
actua soccer 2
ULTIMATE RACE PRO
CREATIVE
WWW.SOUNDBLASTER.COM

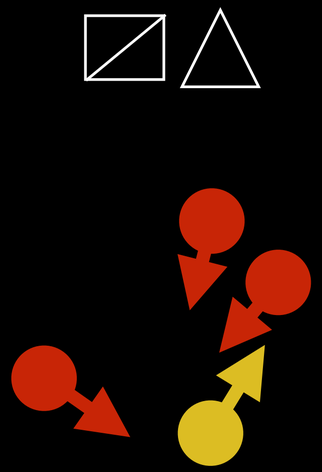

20
0
0
11
108

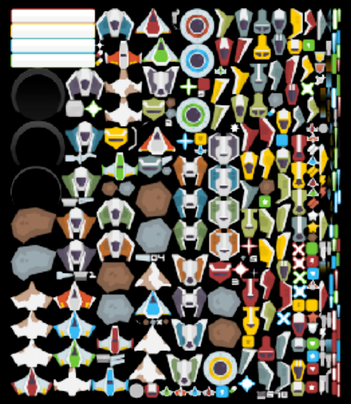

PARK
ONE W

Hardware Rendering

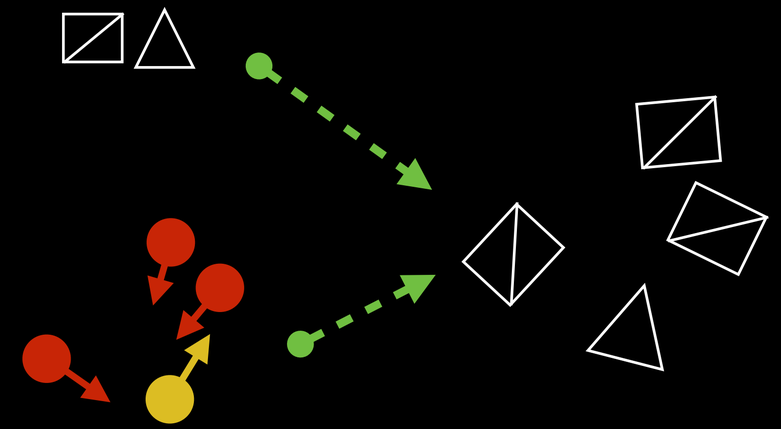
Hardware Rendering

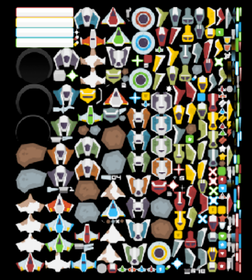
RAM





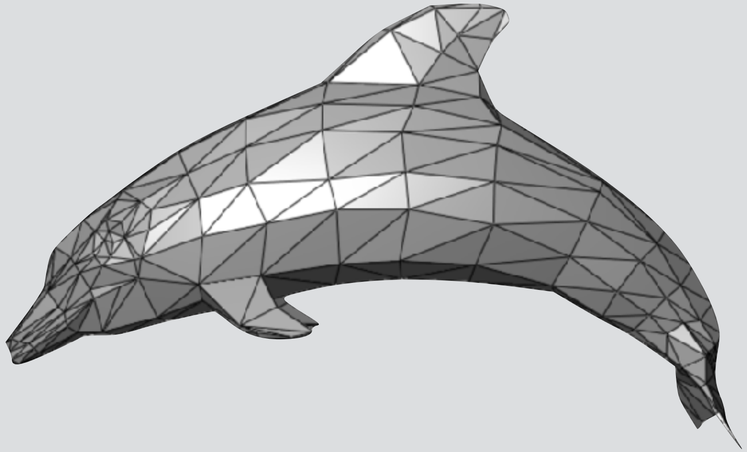
Video card

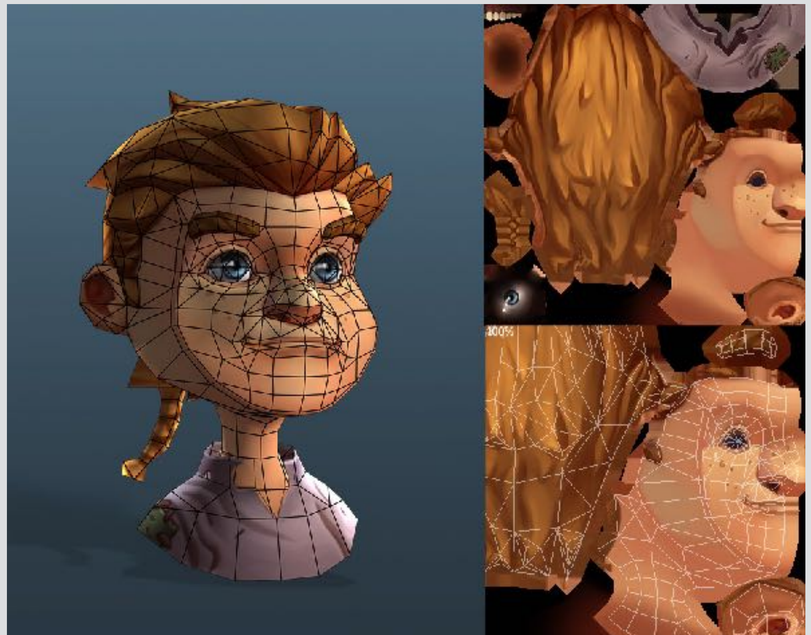




The GPU pipeline

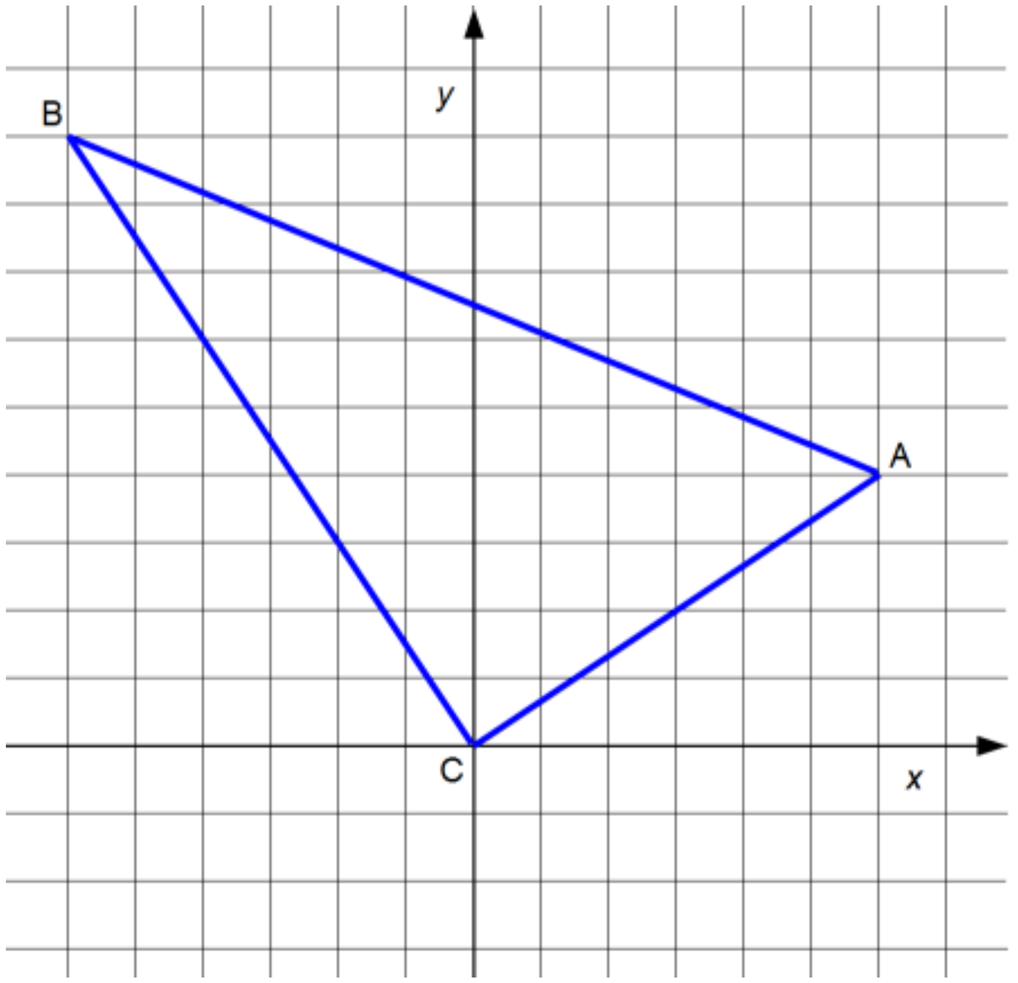
An object is a collection of **polygons**.

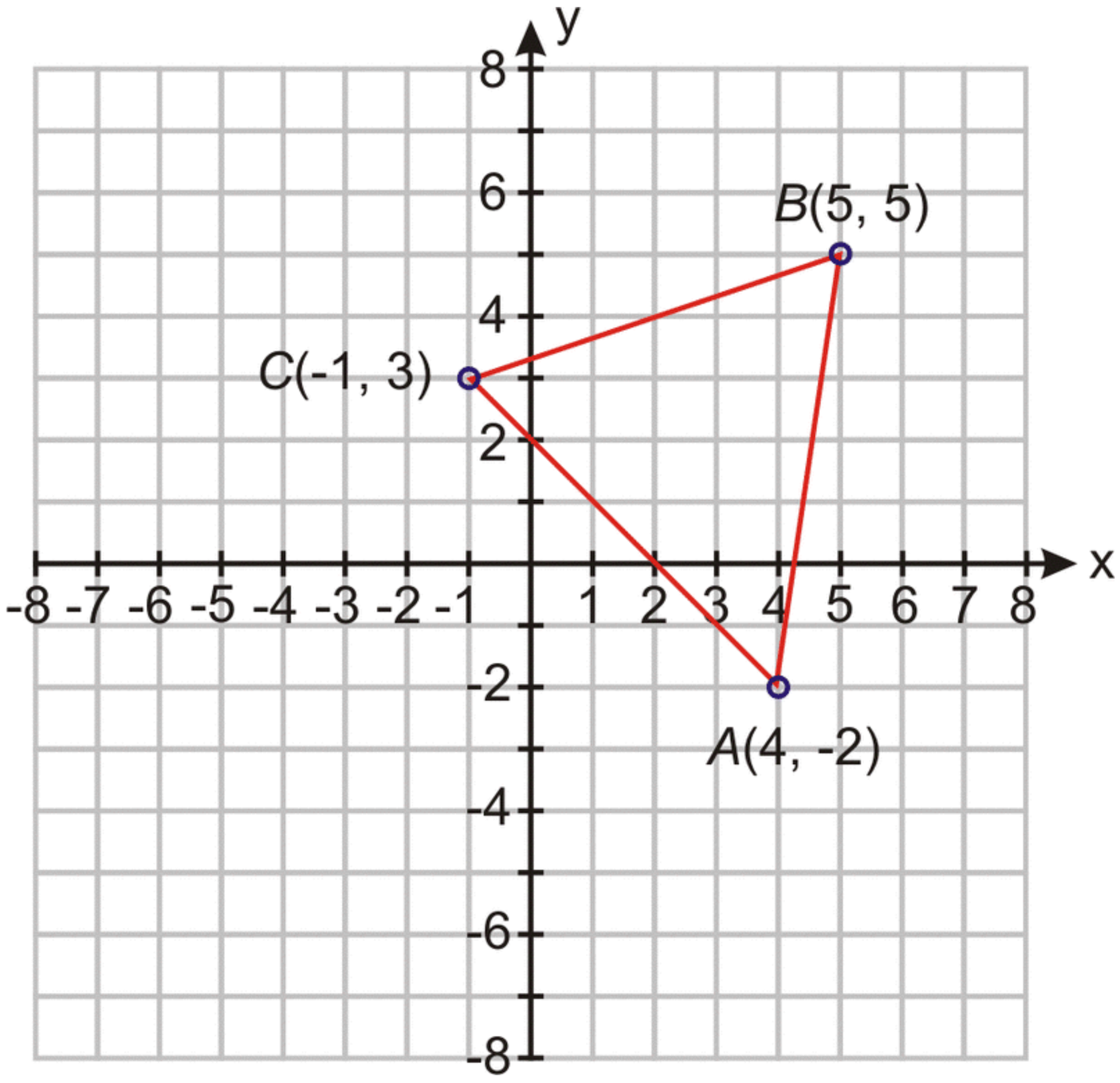


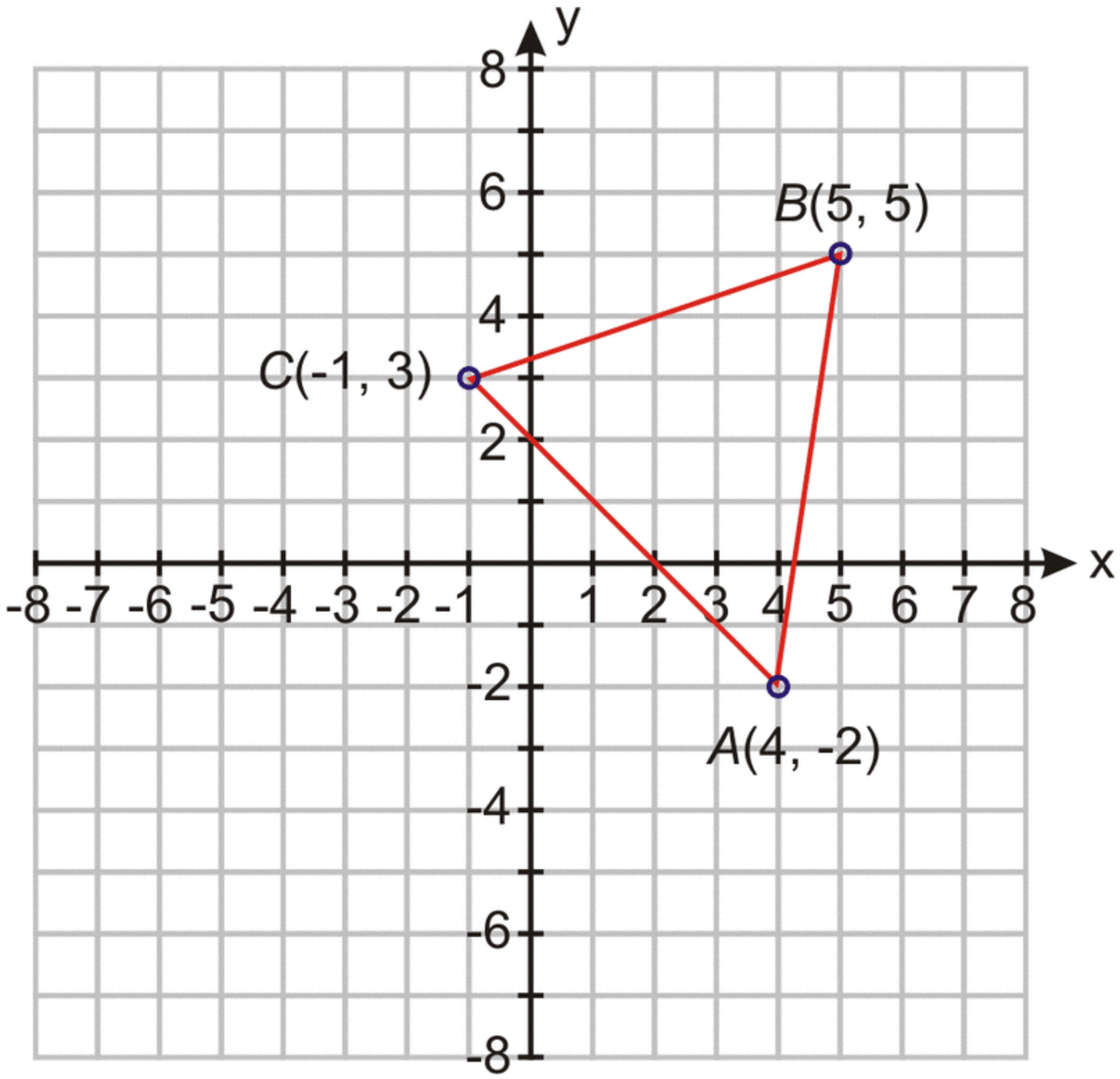


Vertex data

A polygon is defined by
points in space called vertices


B
y
A
C
x


y
x
8
6
4
2
-2
-4
-6
-8
-8 -7 -6 -5 -4 -3 -2 -1
1 2 3 4 5 6 7 8
B(5, 5)
C(-1, 3)
A(4, -2)

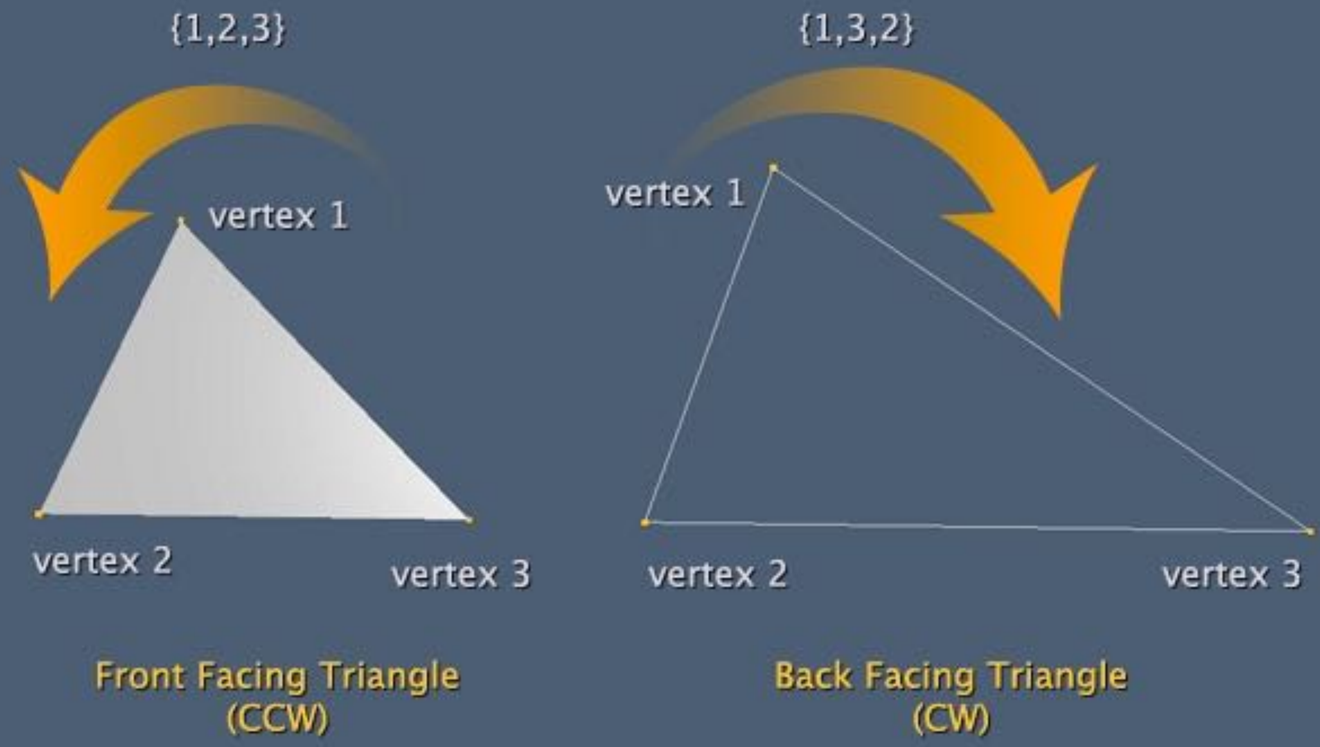


A: (4,−2)

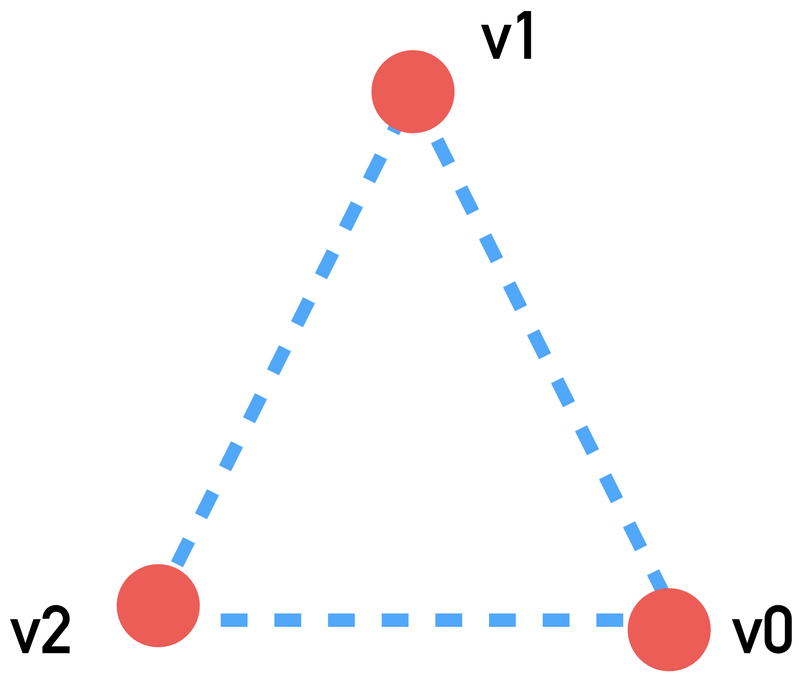
B: (5,5)

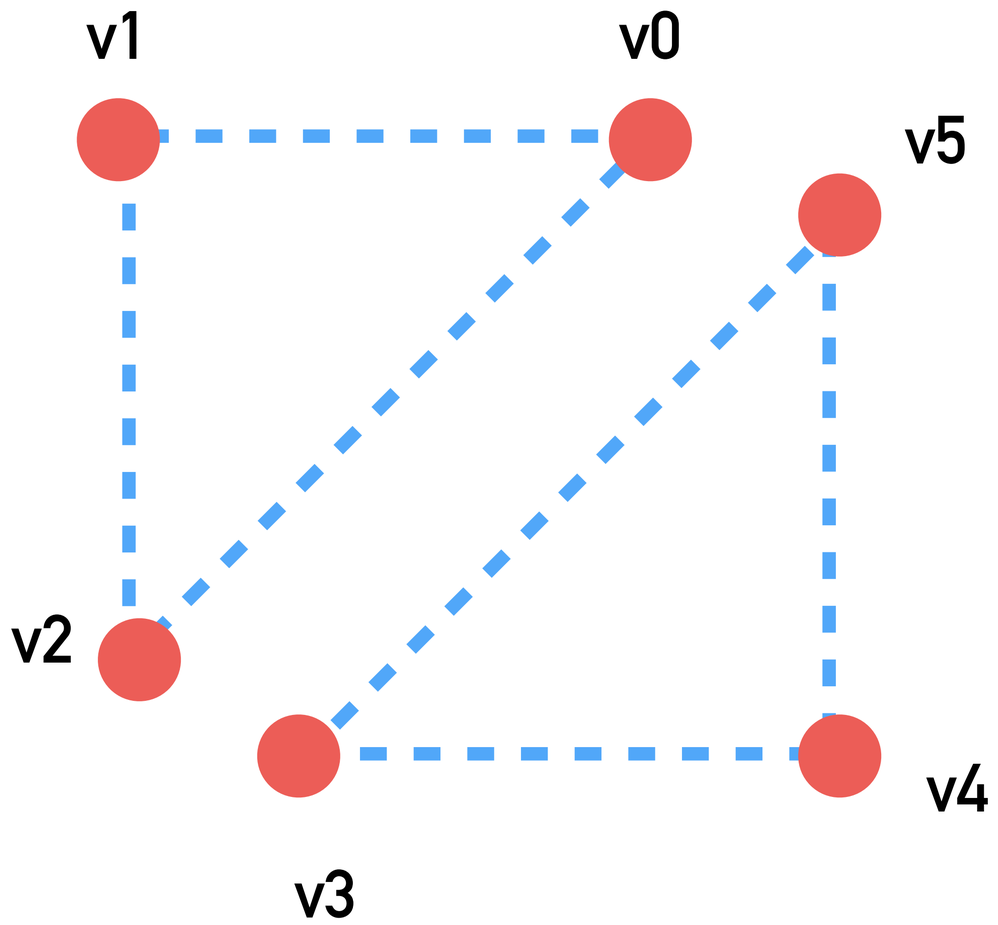
C: (−1, 3)

Polygons are **one-sided** and the side is defined by the **order of the vertices**

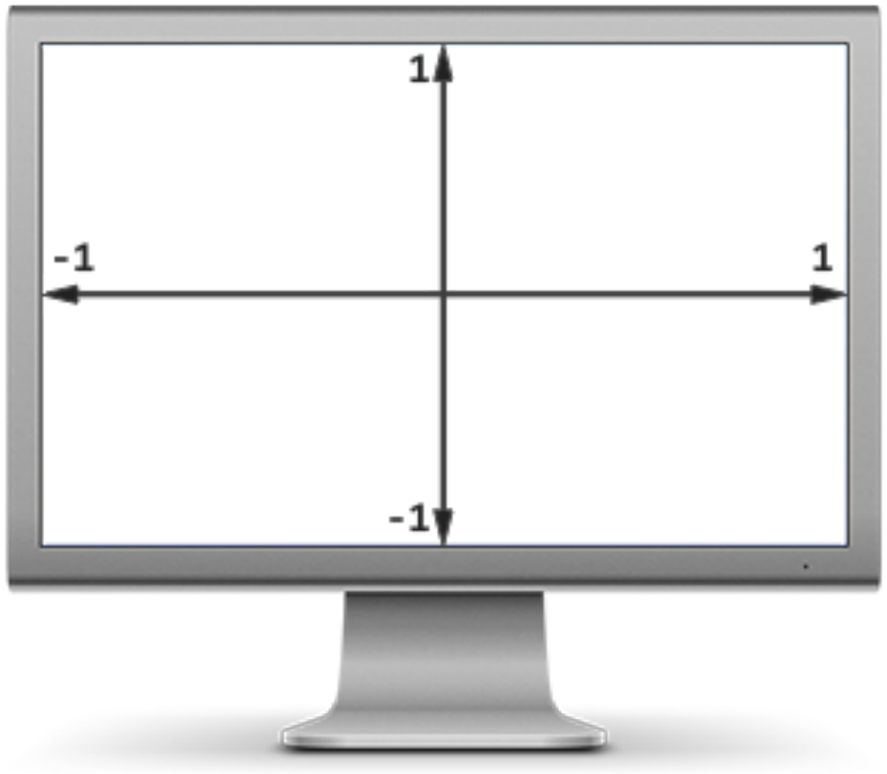


Counter-clockwise order!


v1
v2
v0


v1
v0
v5
v2
v4
v3

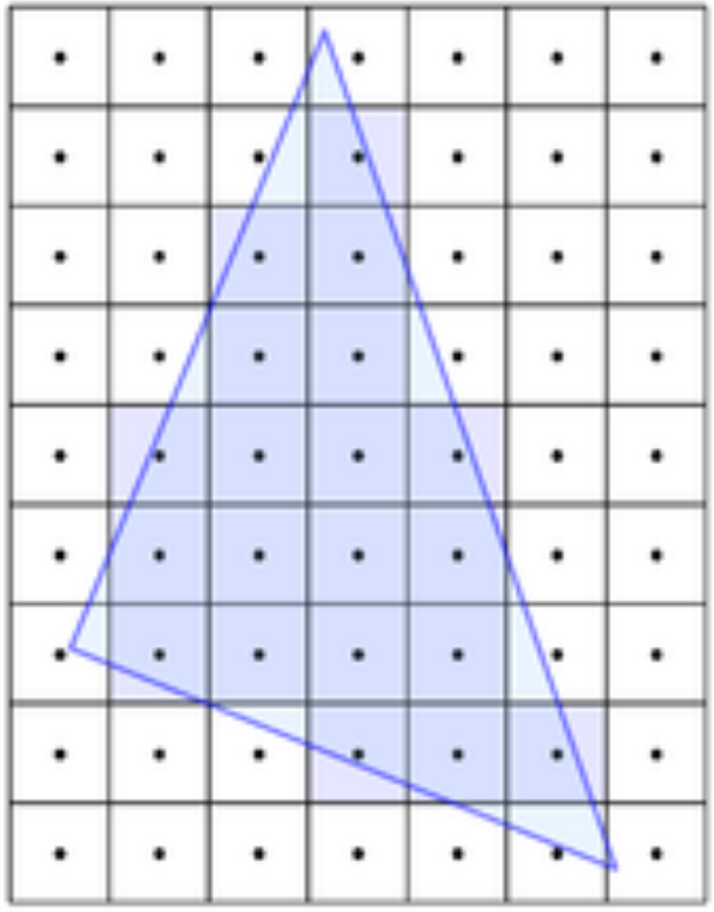
Normalized Device Coordinates



The **GPU** pipeline

VERTEX DATA

Rasterization

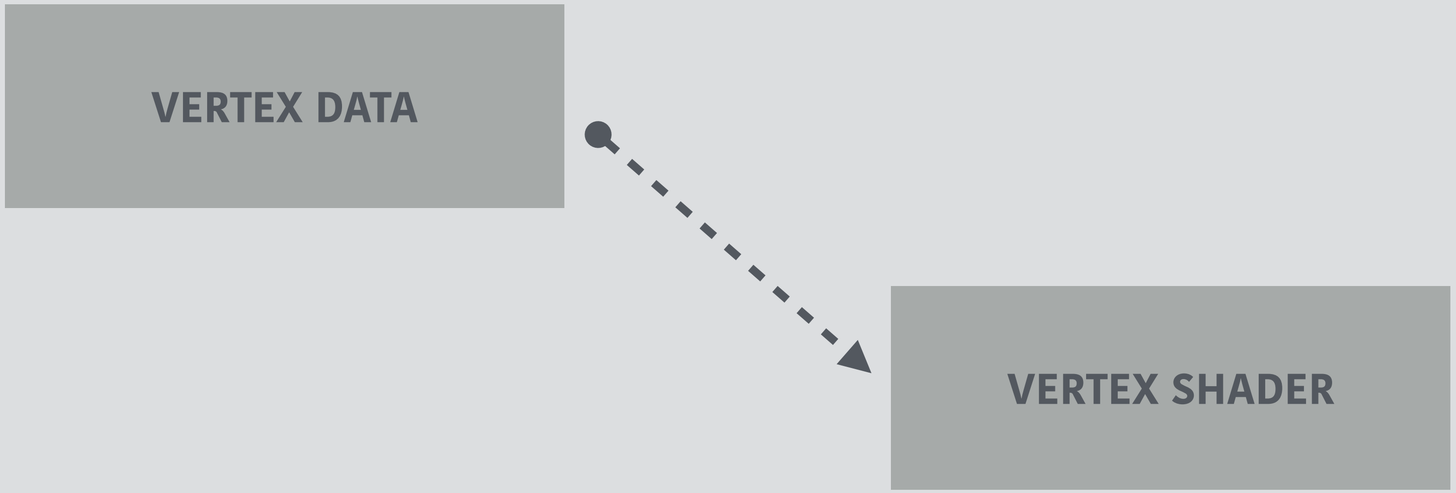


Shaders

The vertex shader

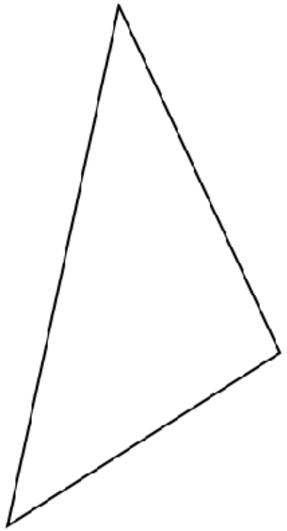
A program that transforms the attributes
(such as position, color or others)
of every vertex passed to the GPU and
returns the final position of the vertex in
device coordinates.

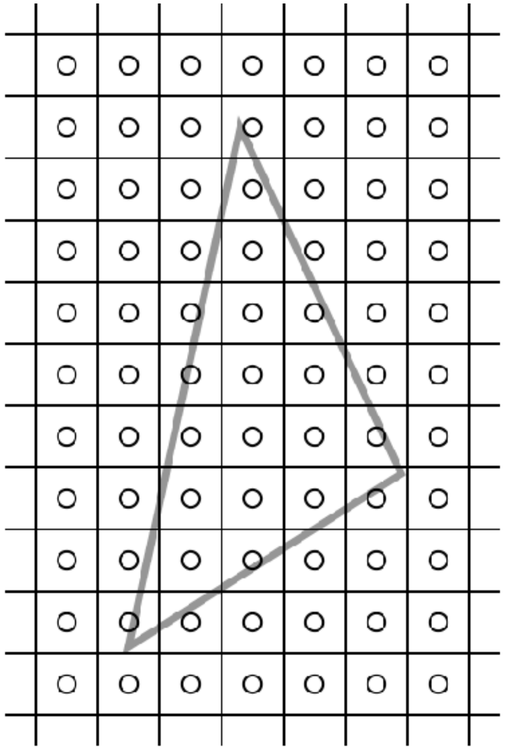
The **GPU** pipeline

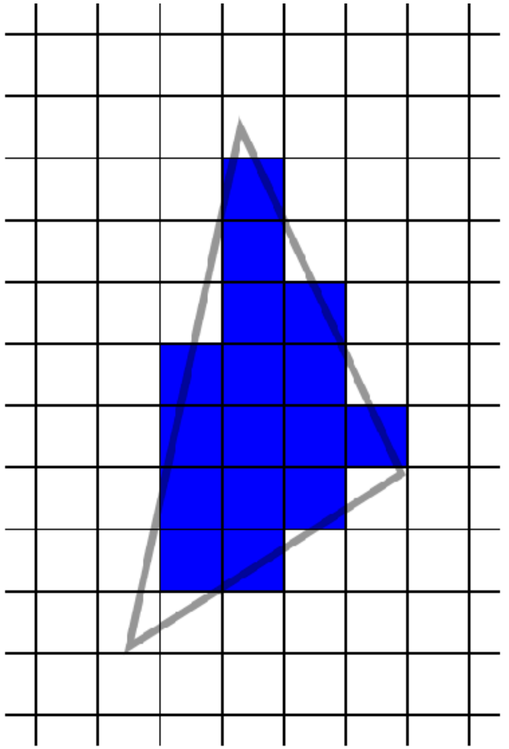


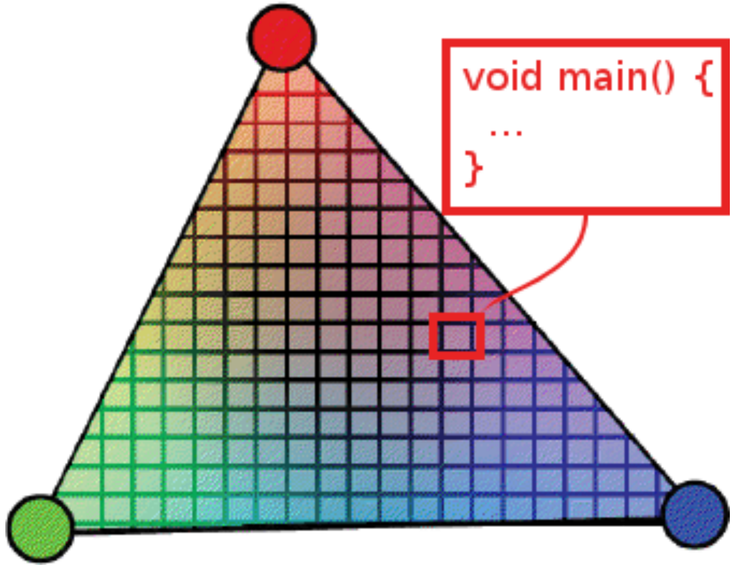
The fragment shader

A **program** that returns **the color of each pixel** when geometry is rasterized on the **GPU**.

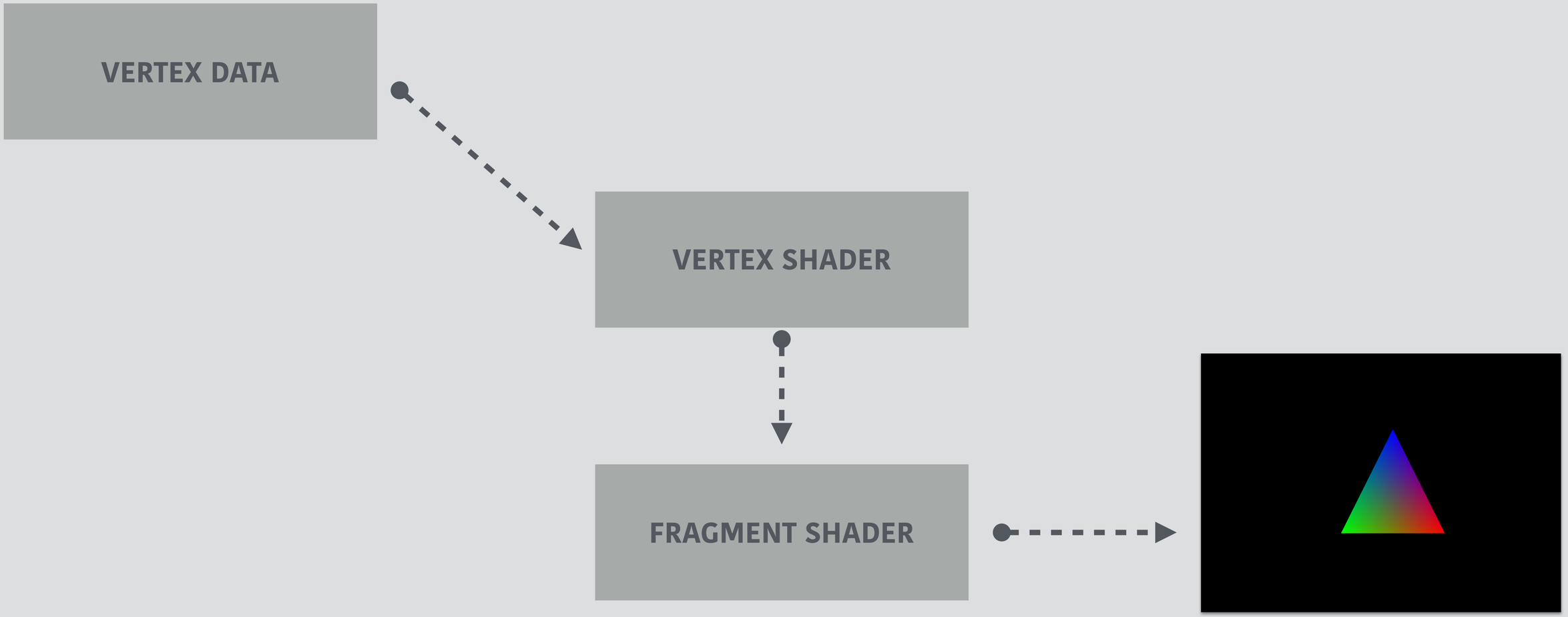







void main() {
...
}

The **GPU** pipeline

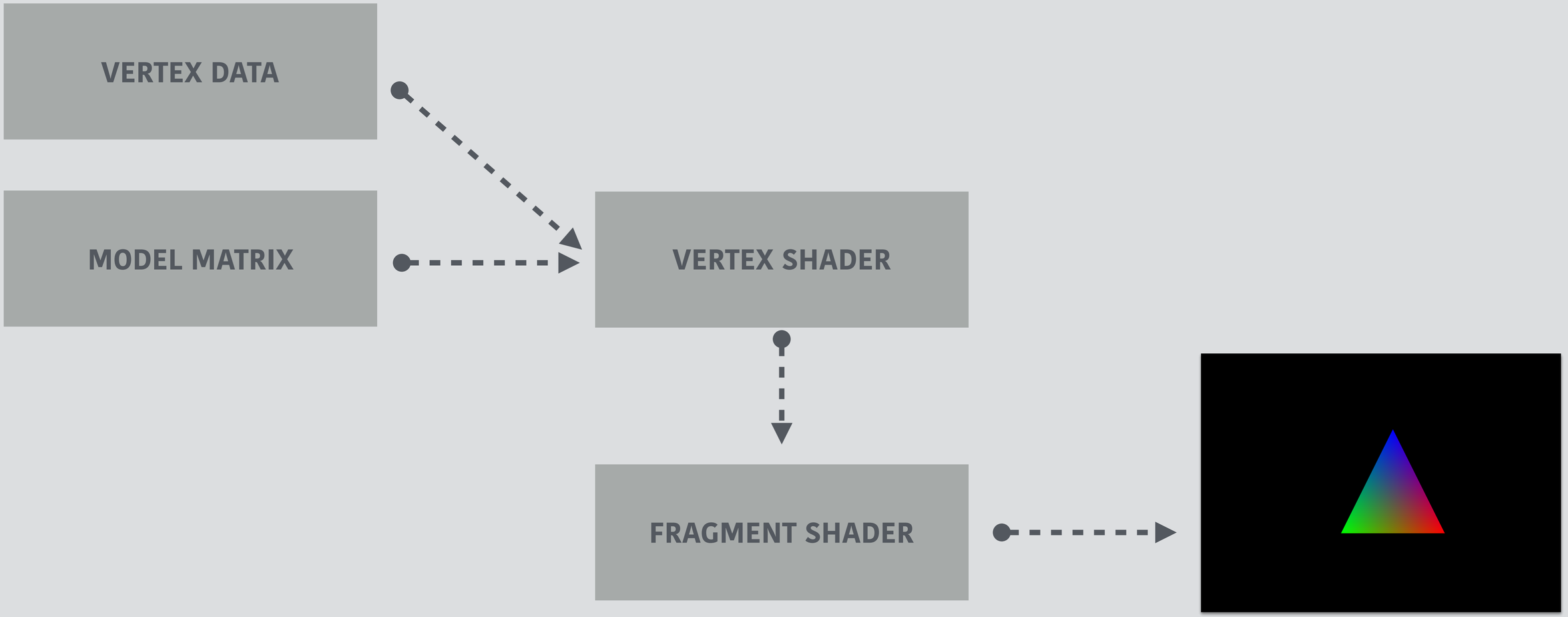


The model matrix

The model matrix is an AFFINE TRANSFORMATION MATRIX
that is multiplied with every vertex you draw.

It is set before drawing each discrete object to
define its transform in space.

The **GPU** pipeline

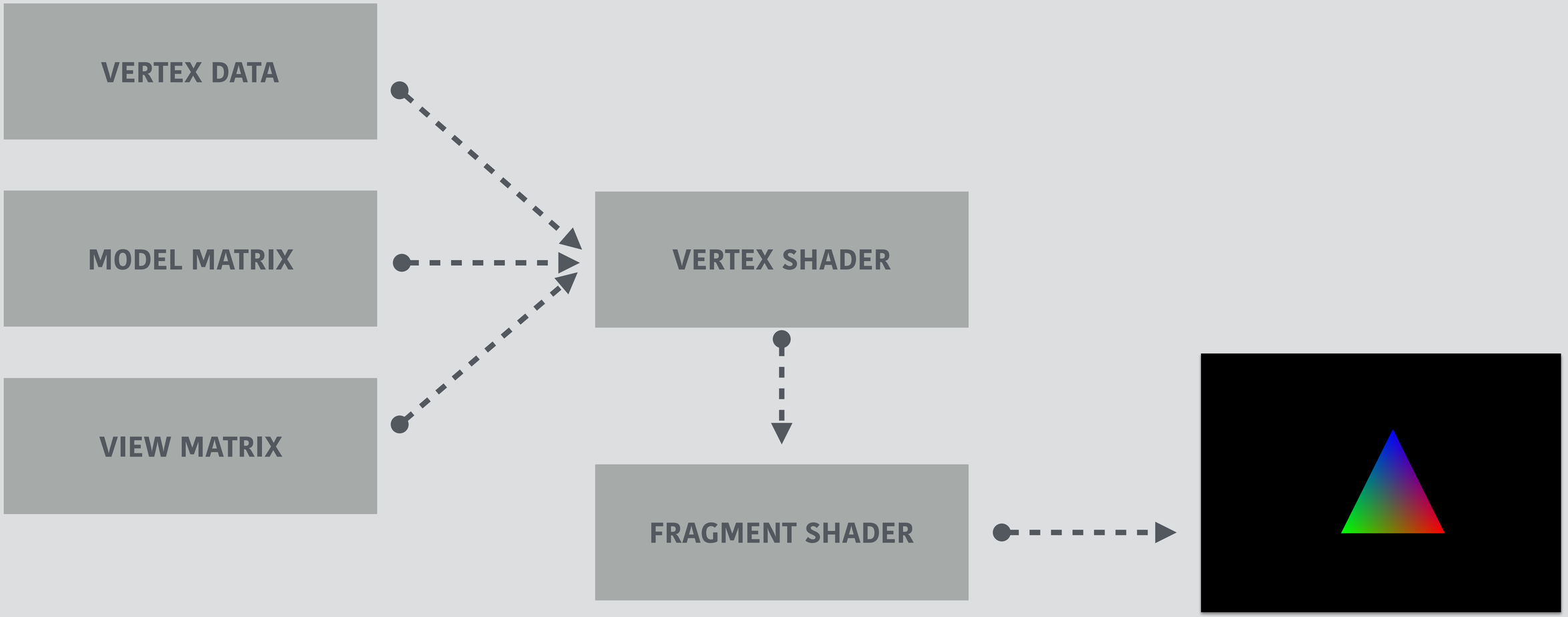


The view matrix

An affine transformation matrix that is applied to the vertices of all objects, letting you transform everything together.

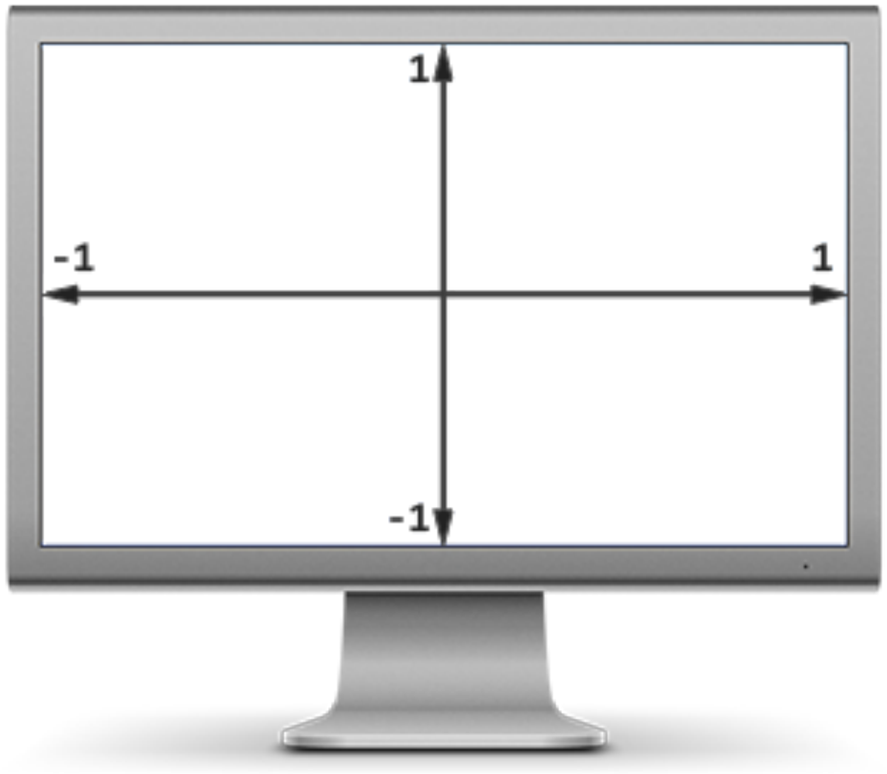
(for now we will keep it identity)

The **GPU** pipeline



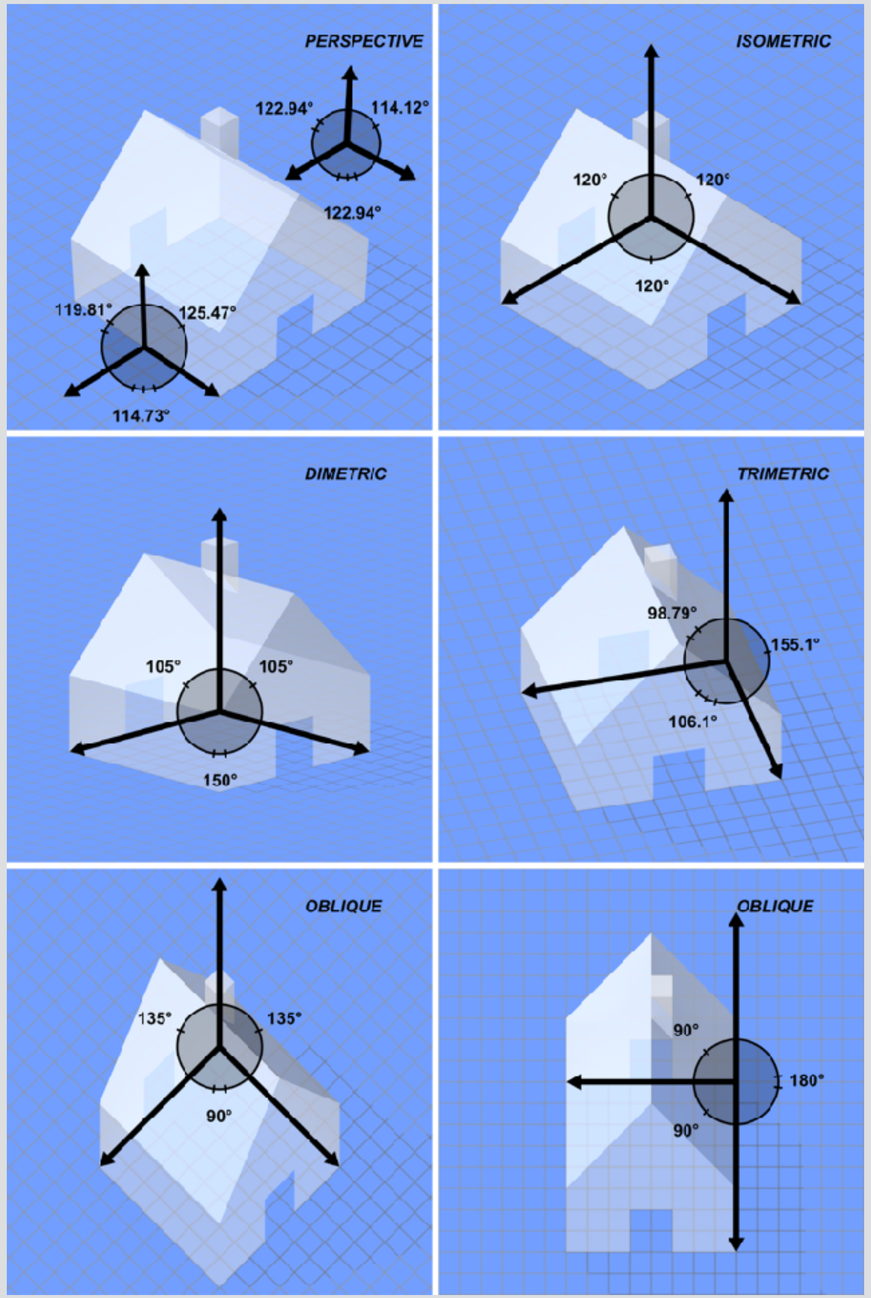
The projection matrix

Normalized Device Coordinates

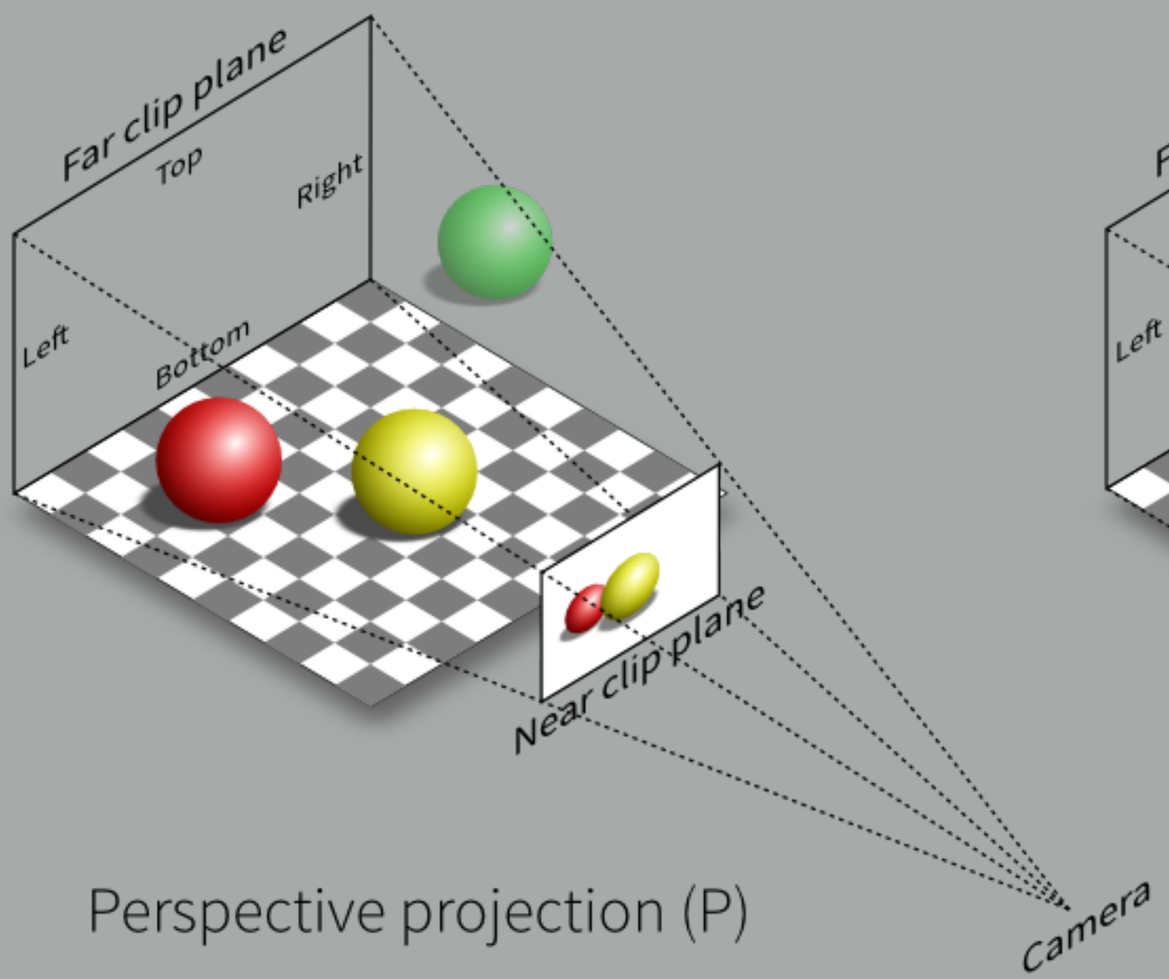


What is projection?

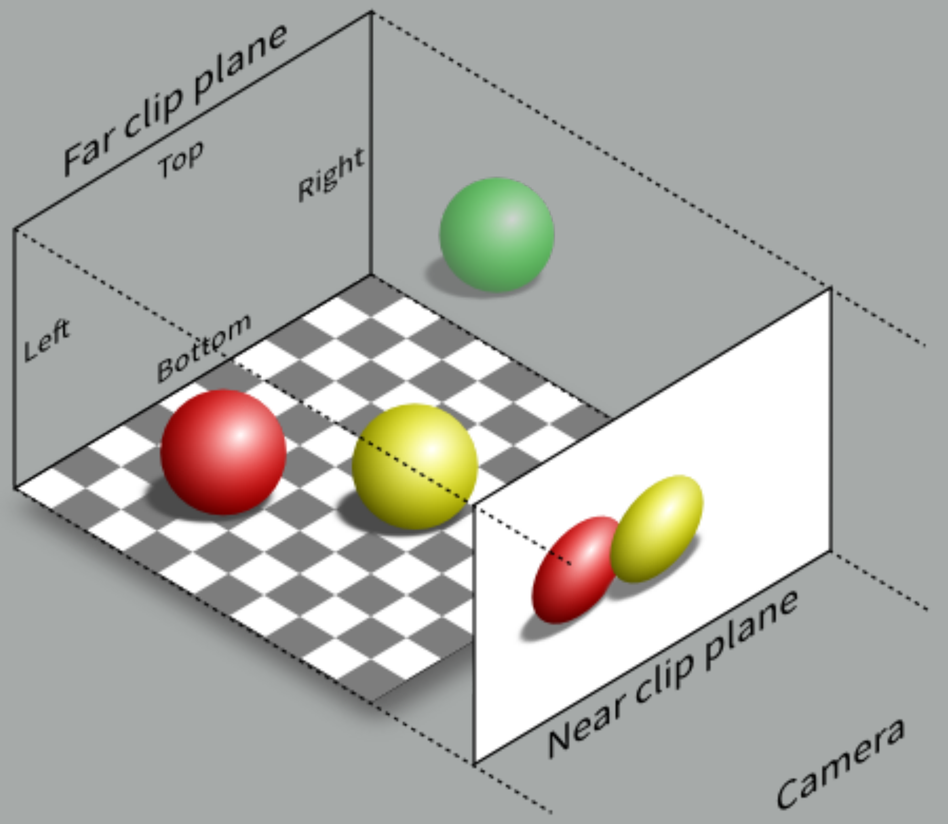
A means of representing a **three-dimensional** object in **two-dimensions**.



Perspective vs. **Orthographic** projection

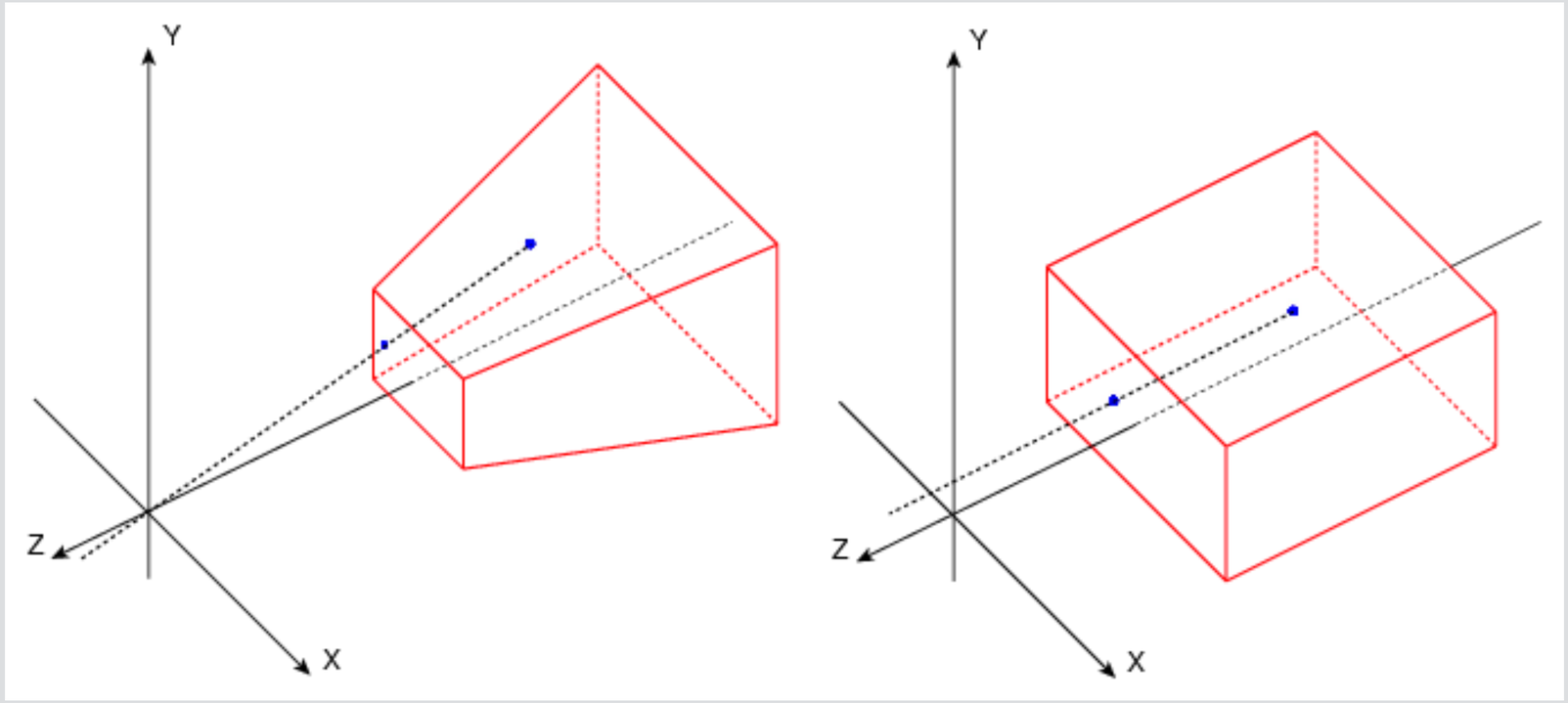


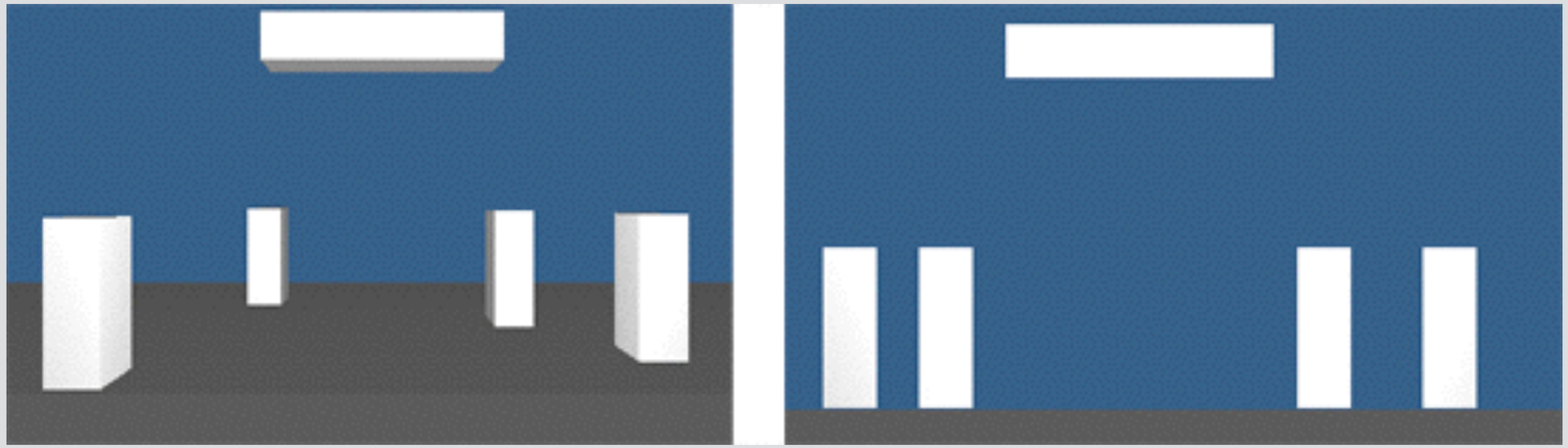
Perspective projection (P)



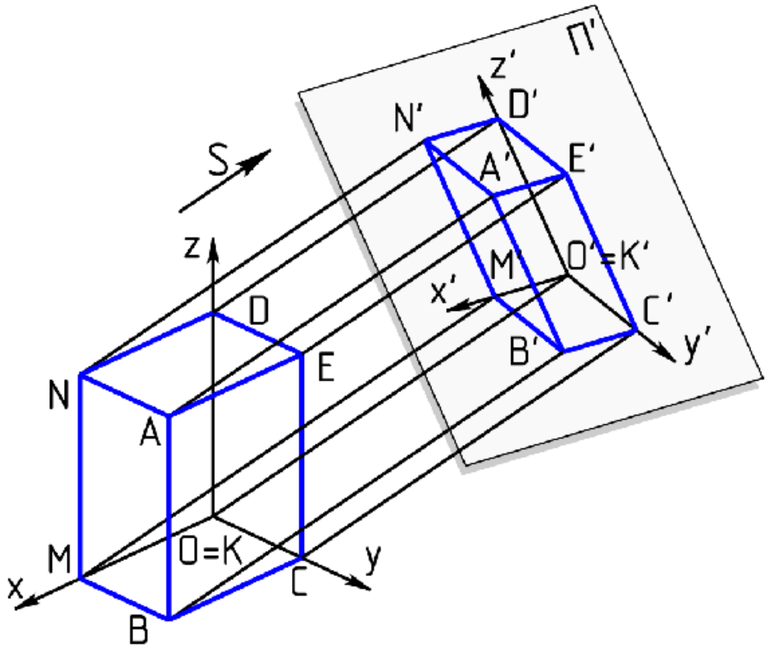
Orthographic projection (O)

Perspective vs. **Orthographic** projection





Projection matrix

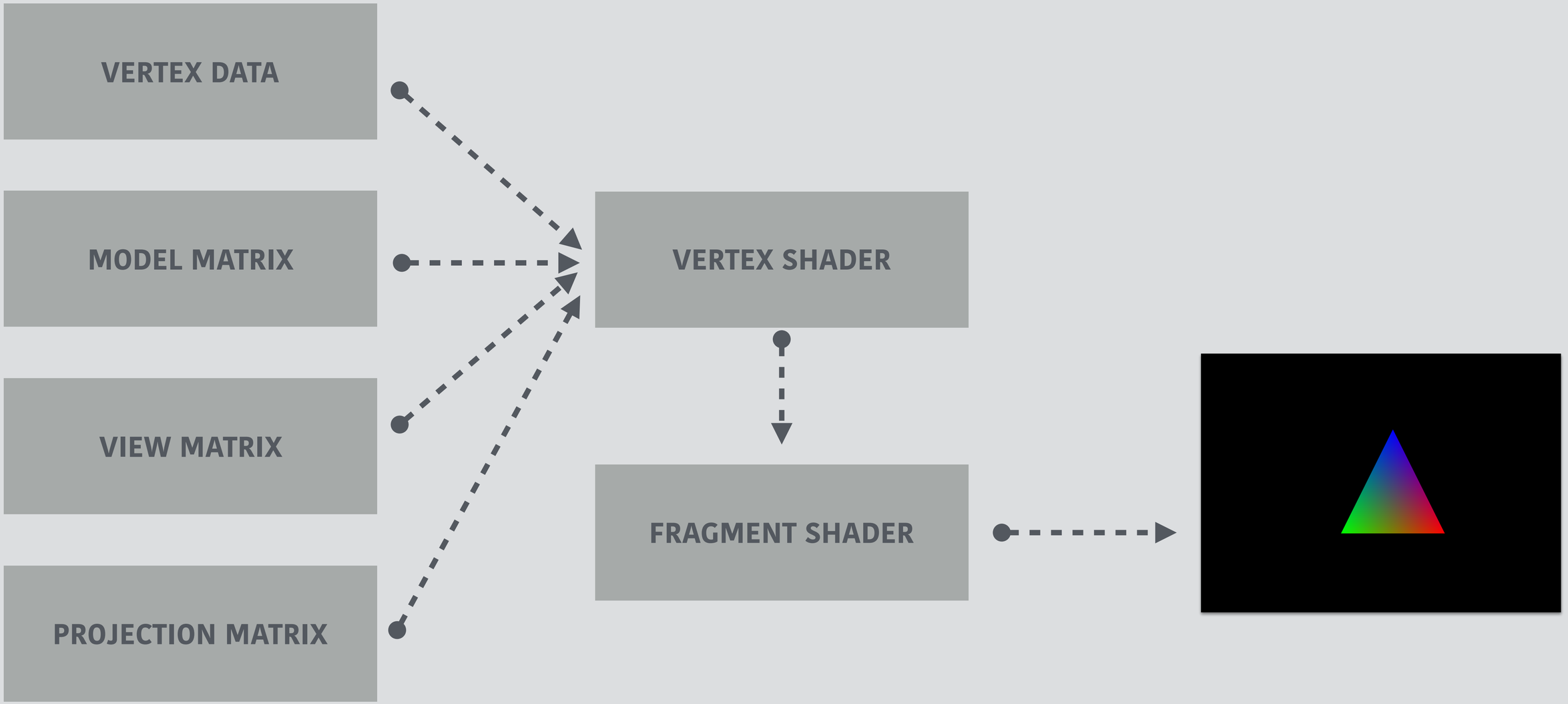


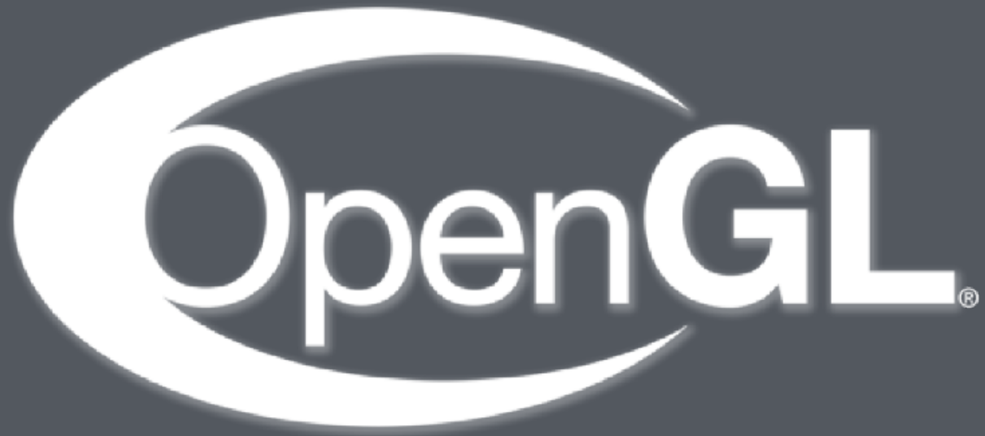
Vertex position * projection matrix = screen vertex position in normalized device coordinates

The projection matrix is a transformation matrix that is multiplied with all vertices to calculate their final position on the screen in normalized device coordinates.

For now we will use orthographic projection since we're doing 2D graphics.

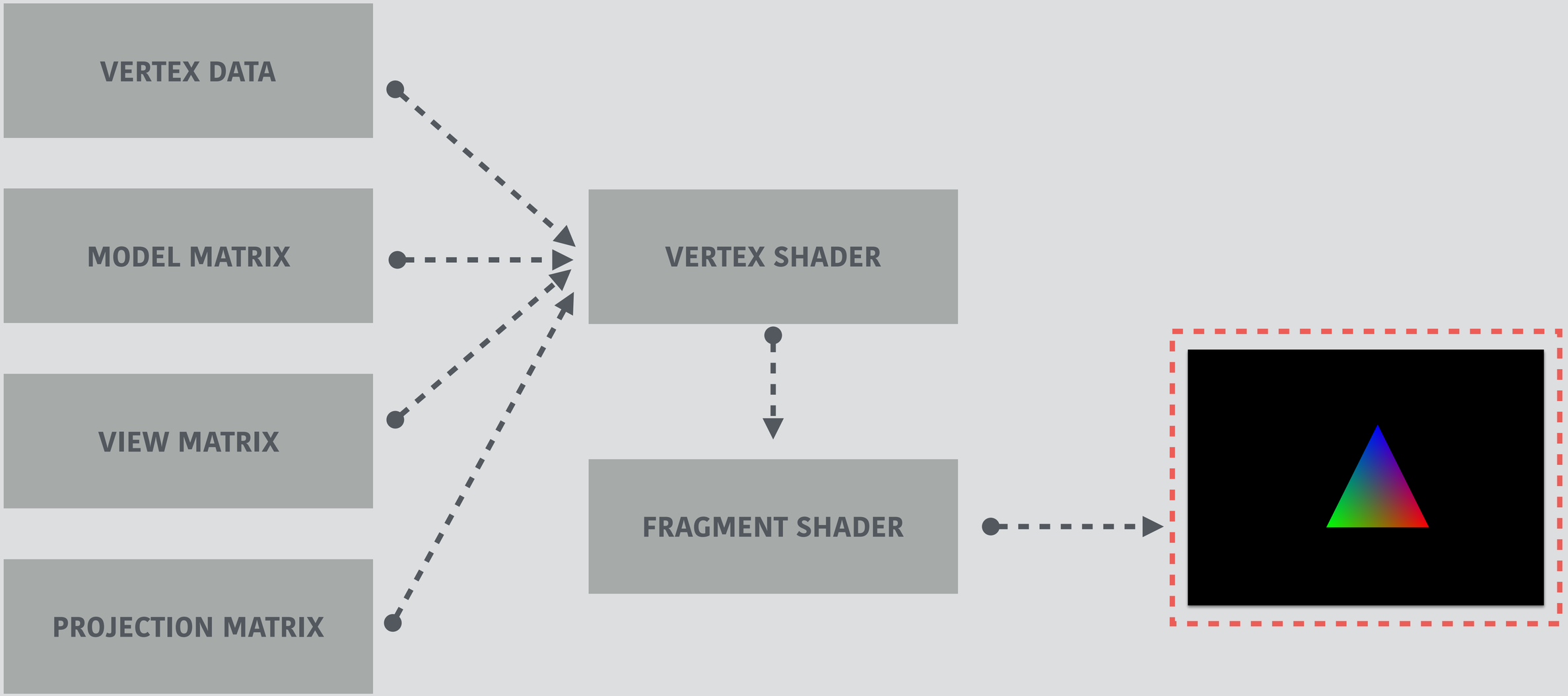
The **GPU** pipeline




OpenGL®

The setup
(happens only once!)

The **GPU** pipeline

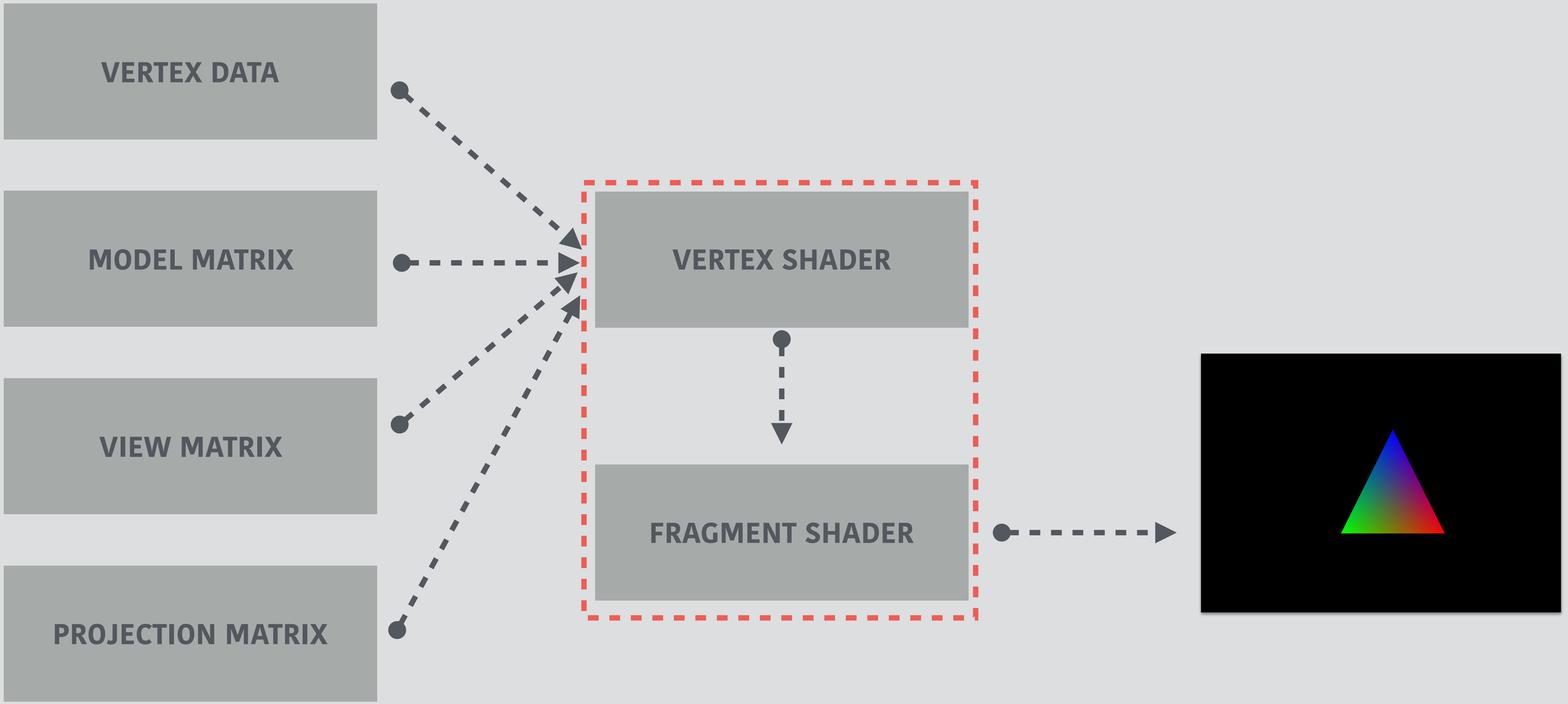


```
void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);
```

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

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

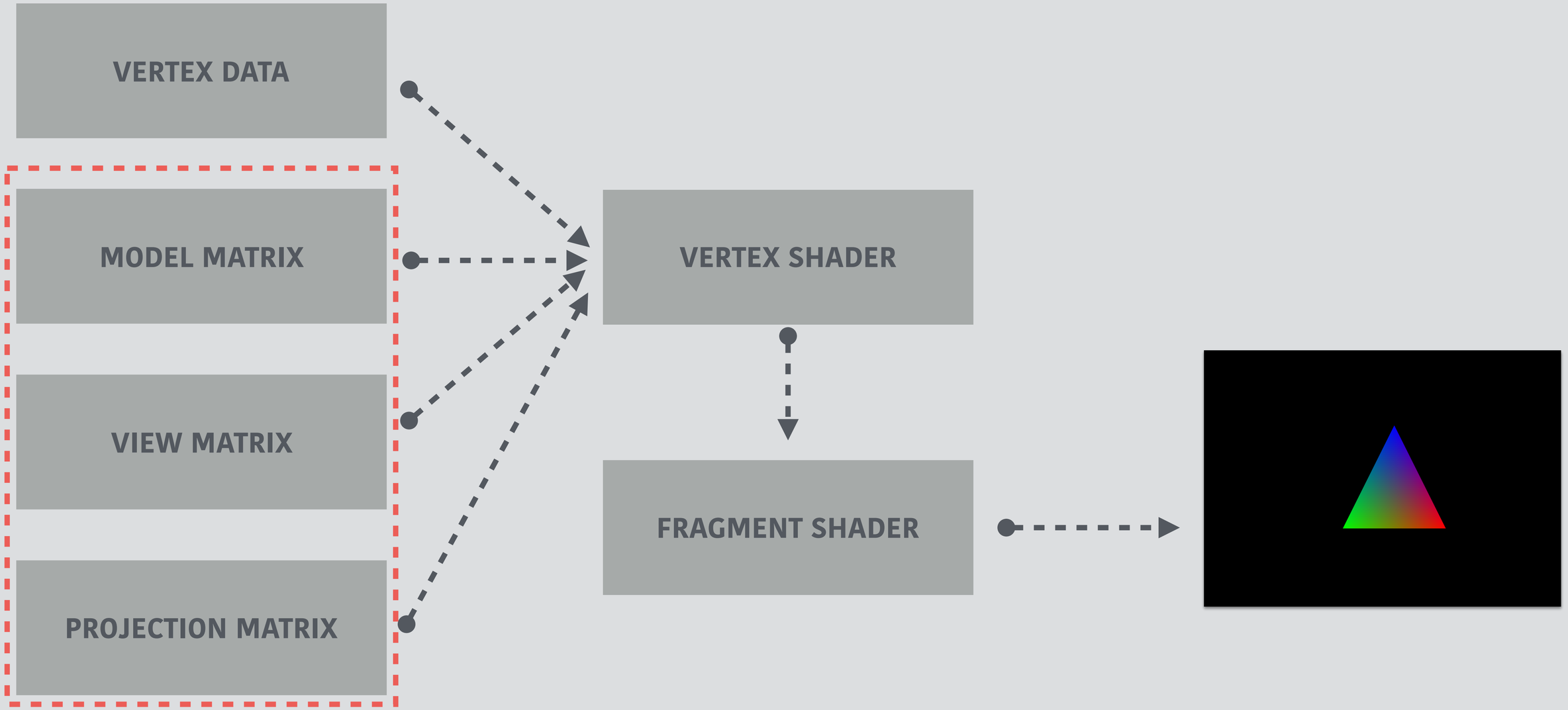
The **GPU** pipeline



The **ShaderProgram** class.

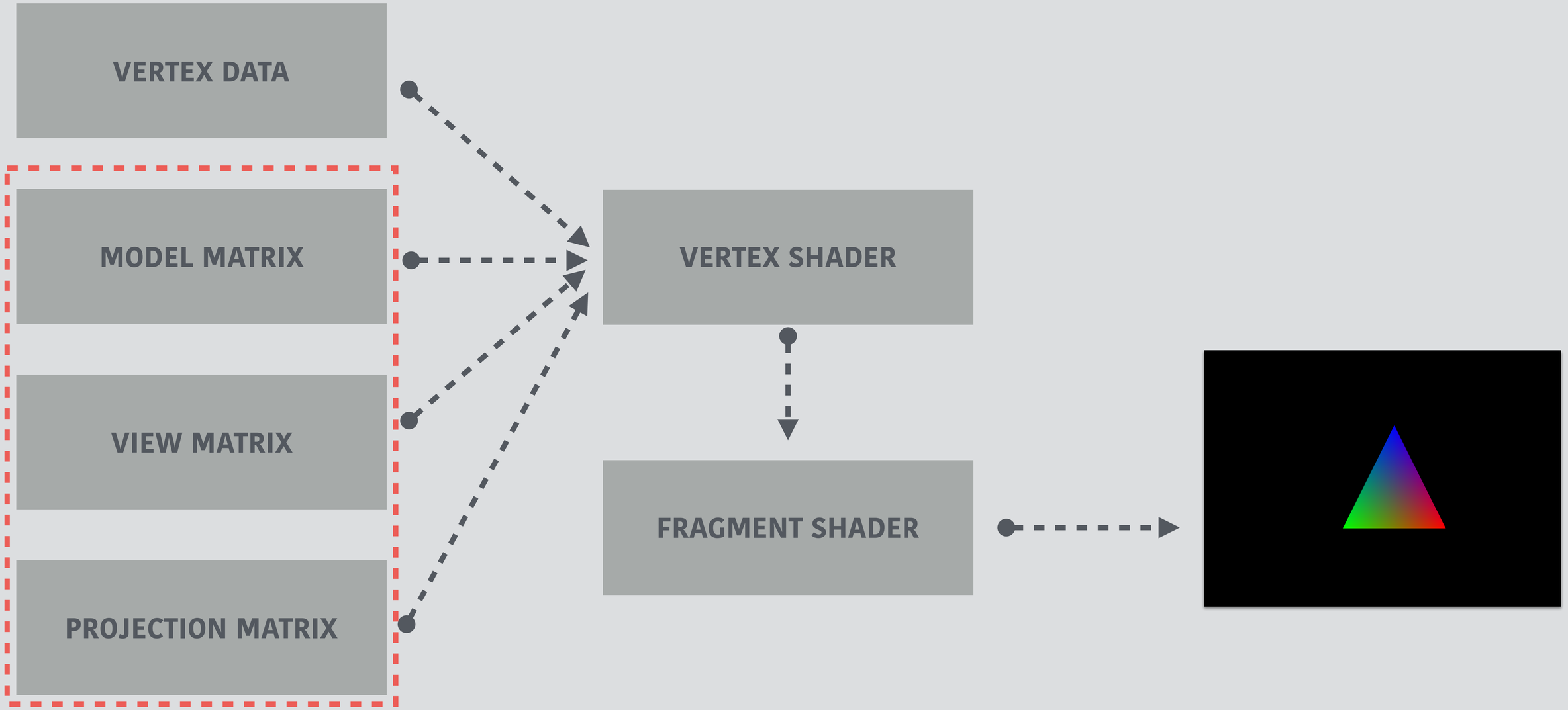
```
#include "ShaderProgram.h"

// …

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

The **GPU** pipeline



GLM math library

The **GPU** pipeline



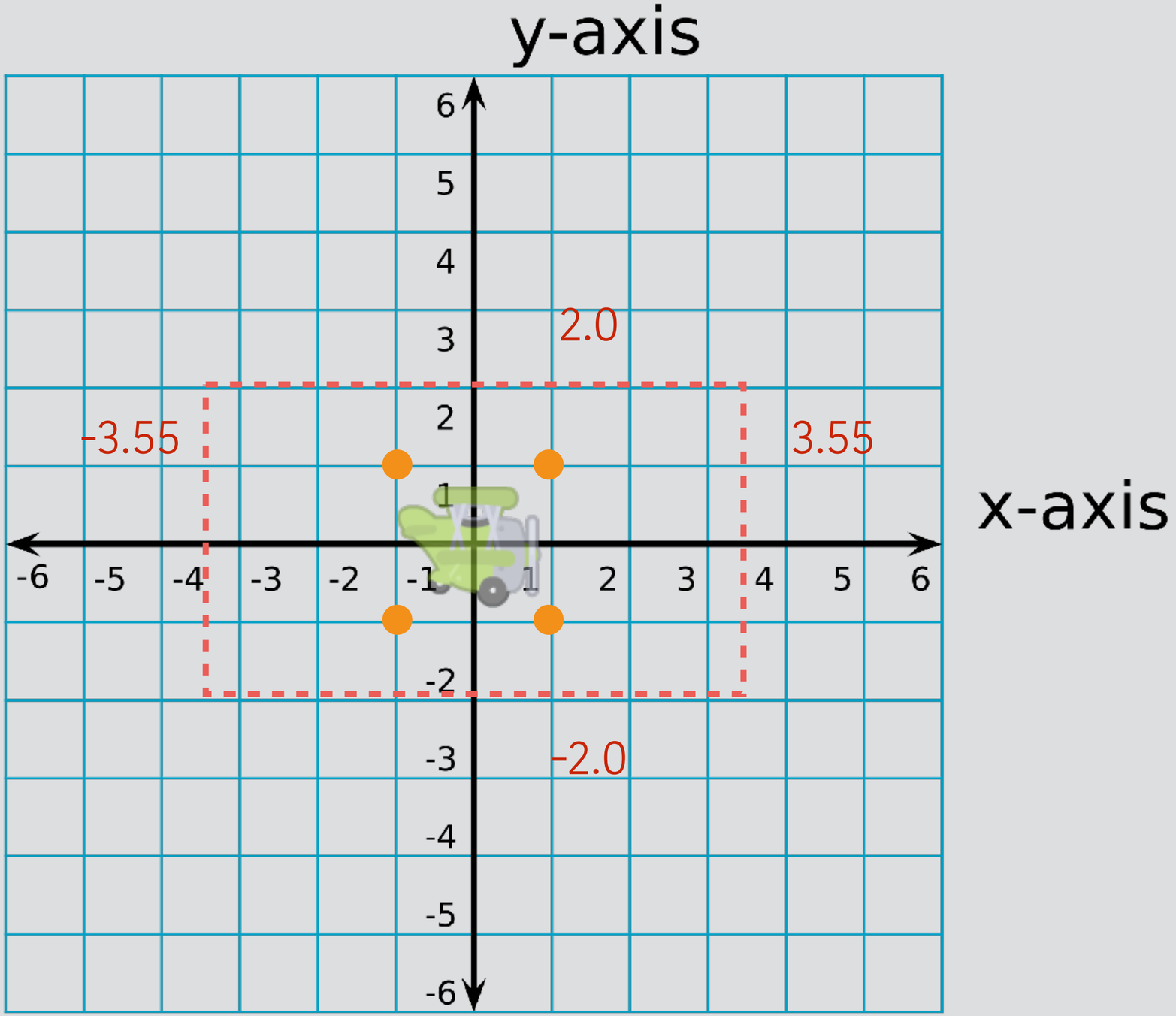
The **Matrix** class.

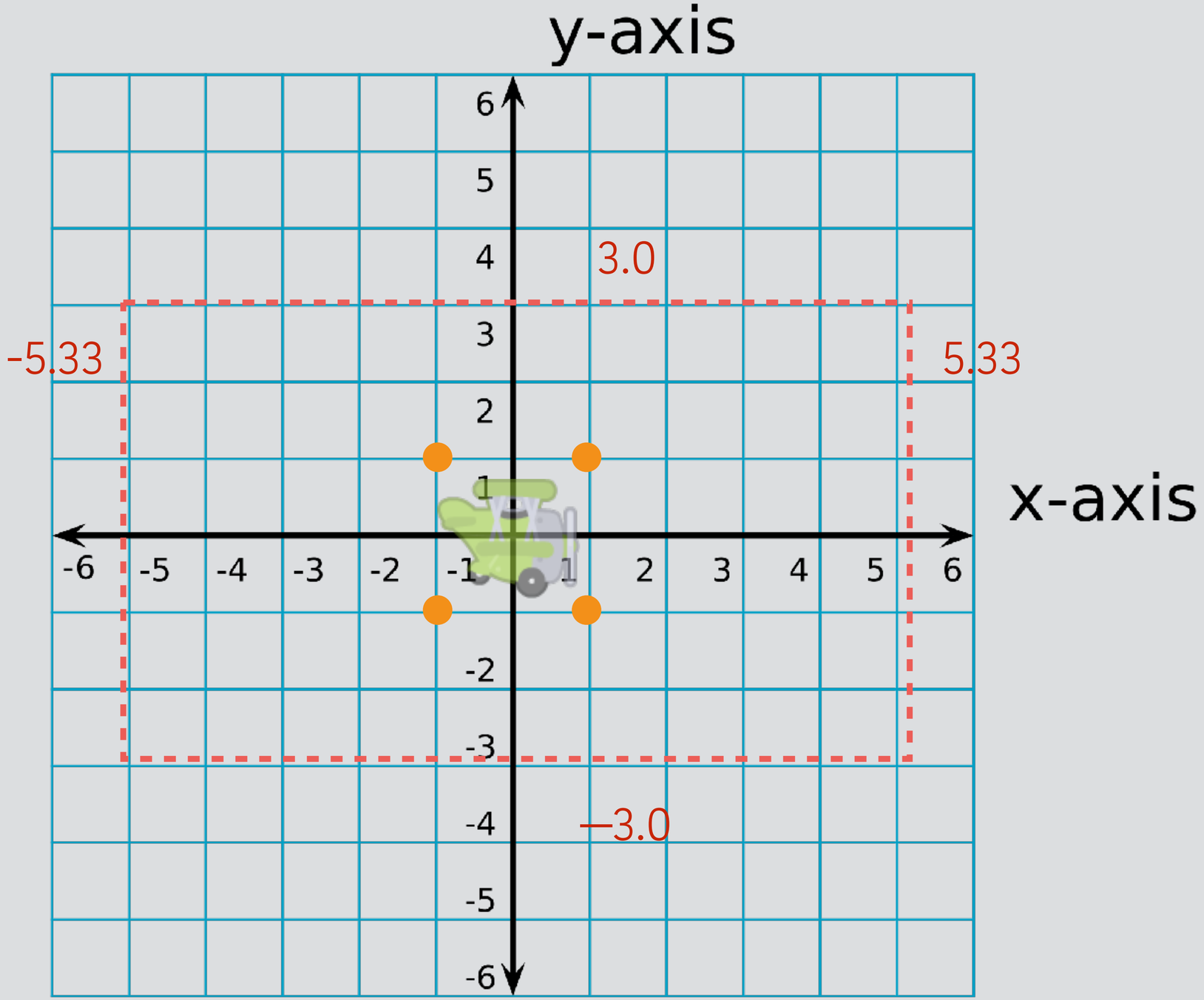
```
#include "glm/mat4x4.hpp"
#include "glm/gtc/matrix_transform.hpp"

// …

glm::mat4 projectionMatrix = glm::mat4(1.0f);
glm::mat4 modelMatrix = glm::mat4(1.0f);
glm::mat4 viewMatrix = glm::mat4(1.0f);
```

```
glm::mat4glm::ortho(float left, float right, float bottom,
float top, float near, float far);
```

Sets an orthographic projection in a matrix.

```
glm::mat4 projectionMatrix = glm::mat4(1.0f);
projectionMatrix = glm::ortho(-1.777f, 1.777f, -1.0f, 1.0f, -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

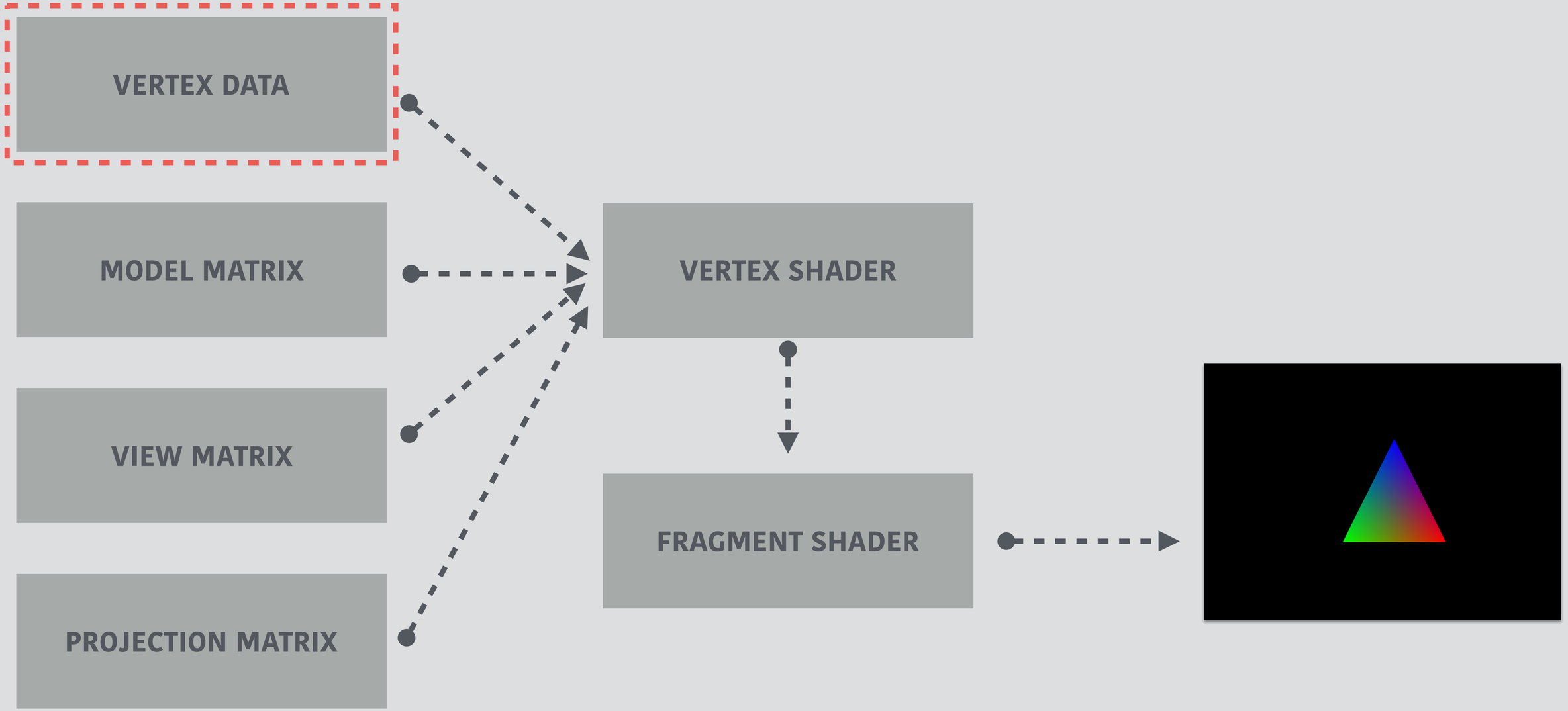

y-axis
x-axis
3.0
-3.0
-5.33
5.33
6
5
4
3
2
1
-1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

Drawing polygons.
(happens every frame!)

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

```
program.SetModelMatrix(modelMatrix);
program.SetProjectionMatrix(projectionMatrix);
program.SetViewMatrix(viewMatrix);
```

The **GPU** pipeline



```
void glUseProgram (GLint programID);
```

Use the specified **program id**.

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

```
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);
```

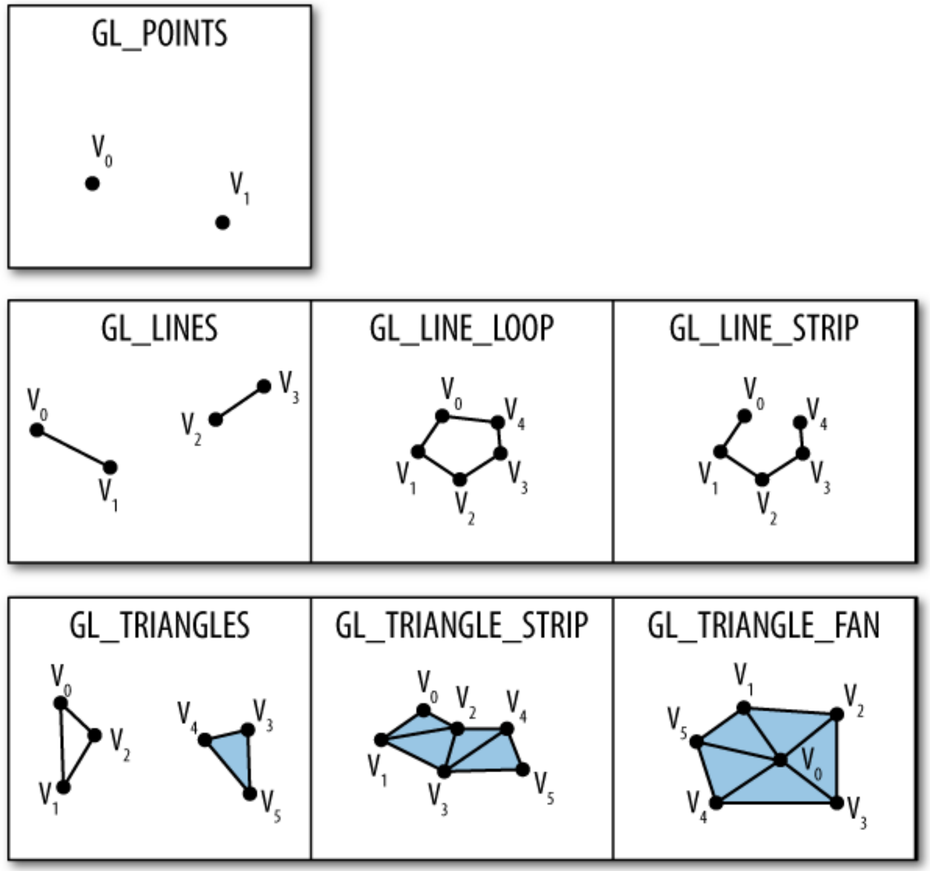
Enables a **vertex attribute**.

```
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.

Setup (before the loop)

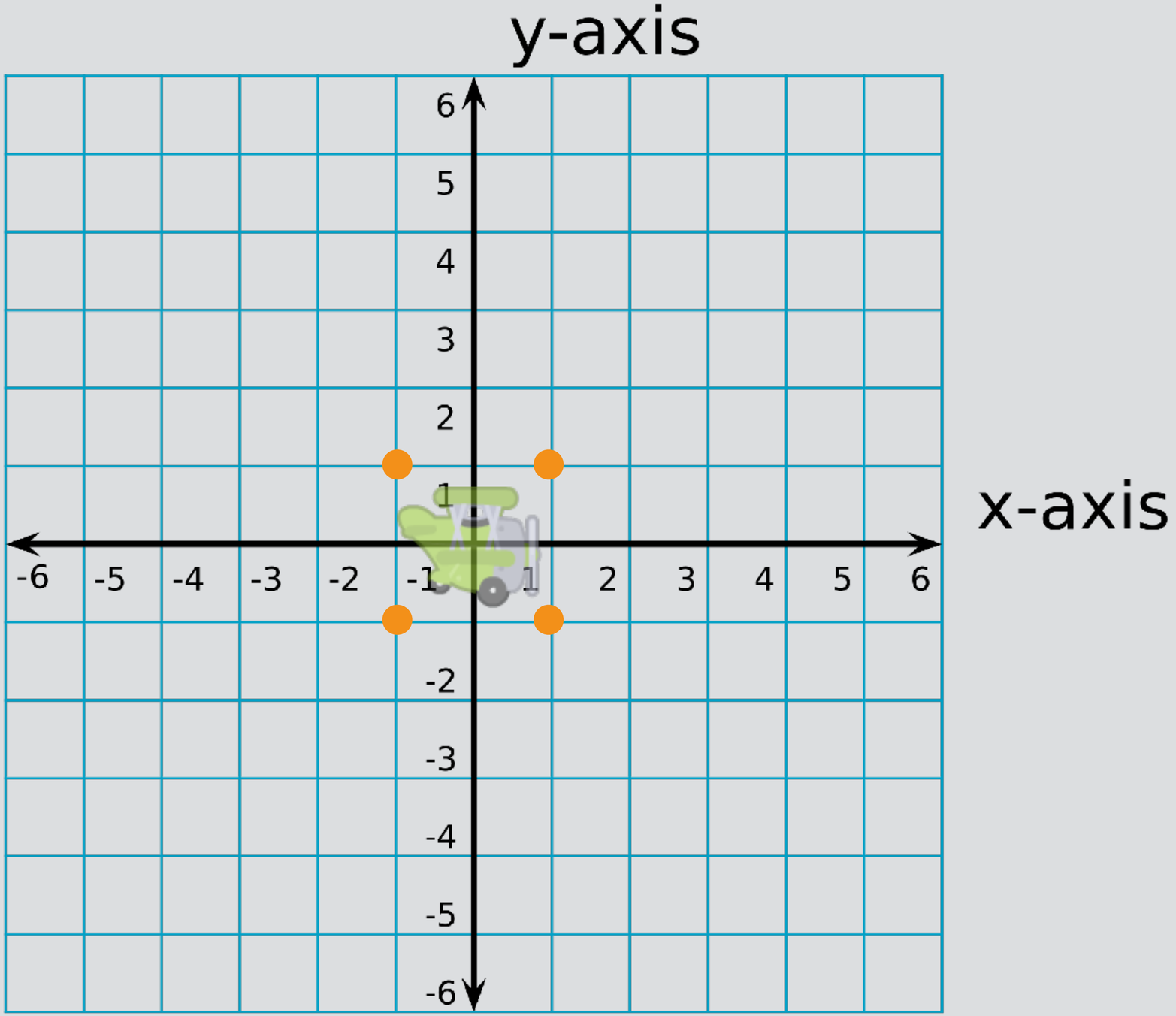
```
glViewport(0, 0, 640, 360);

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

glm::mat4 projectionMatrix = glm::mat4(1.0f);
glm::mat4 modelMatrix = glm::mat4(1.0f);
glm::mat4 viewMatrix = glm::mat4(1.0f);

projectionMatrix = glm::ortho(-1.777f, 1.777f, -1.0f, 1.0f, -1.0f, 1.0f);

glUseProgram(program.programID);
```

Drawing (in your game loop)

```
glClear(GL_COLOR_BUFFER_BIT);

program.SetModelMatrix(modelMatrix);
program.SetProjectionMatrix(projectionMatrix);
program.SetViewMatrix(viewMatrix);

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.

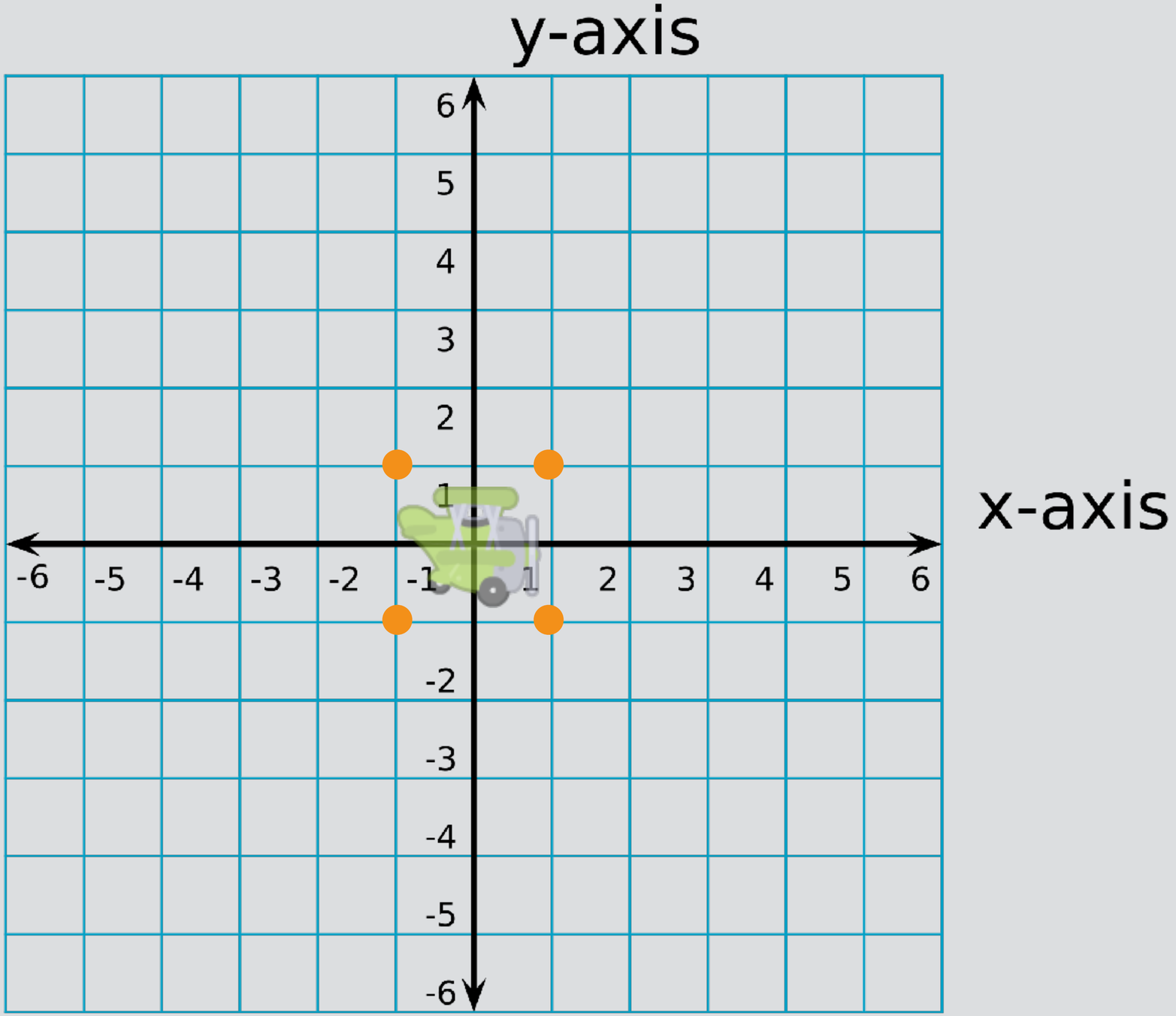

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

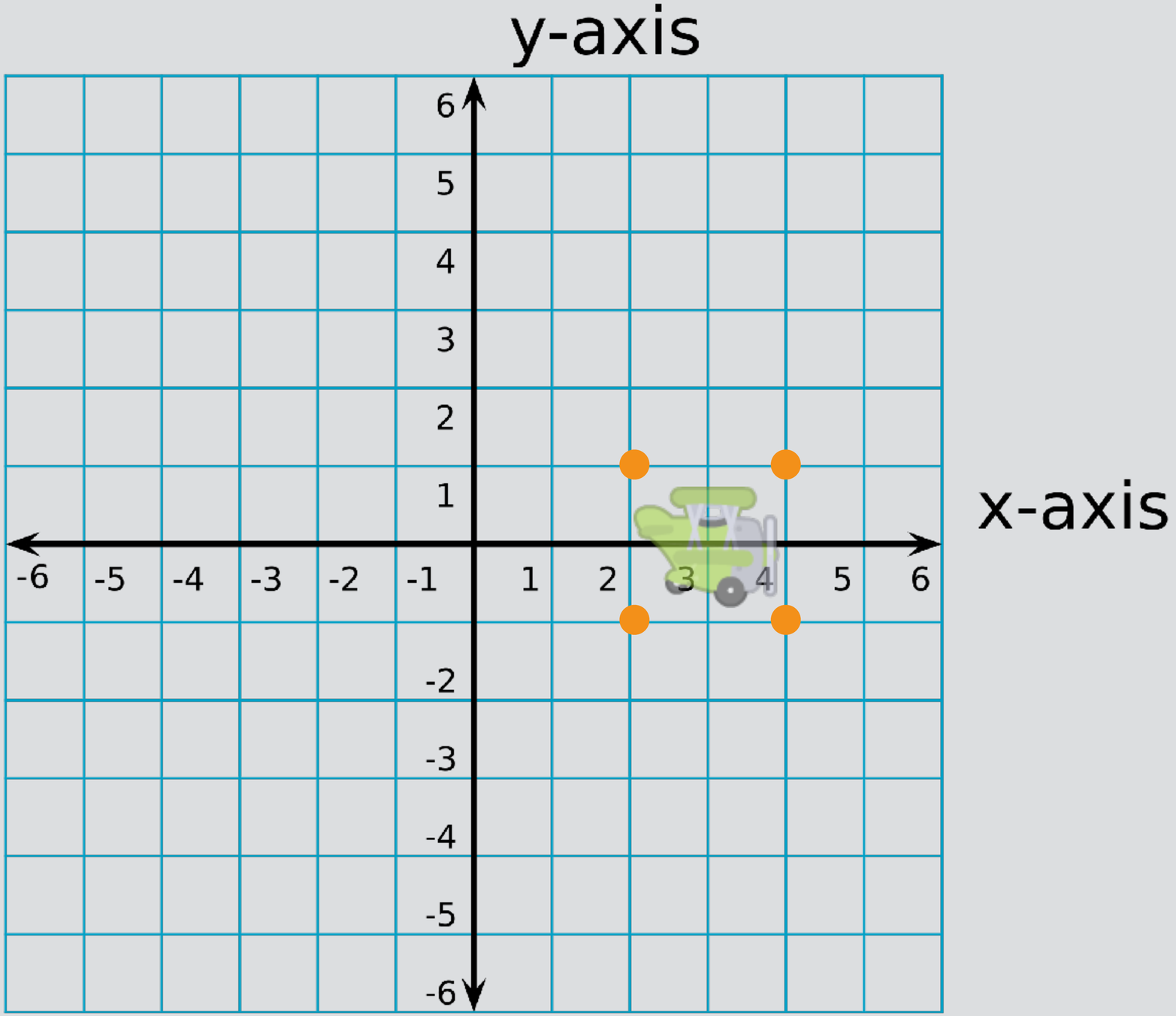
`glm::mat4(1.0f)`

Creates an identity matrix.

```
glm::mat4 modelMatrix = glm::mat4(1.0f);
```

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

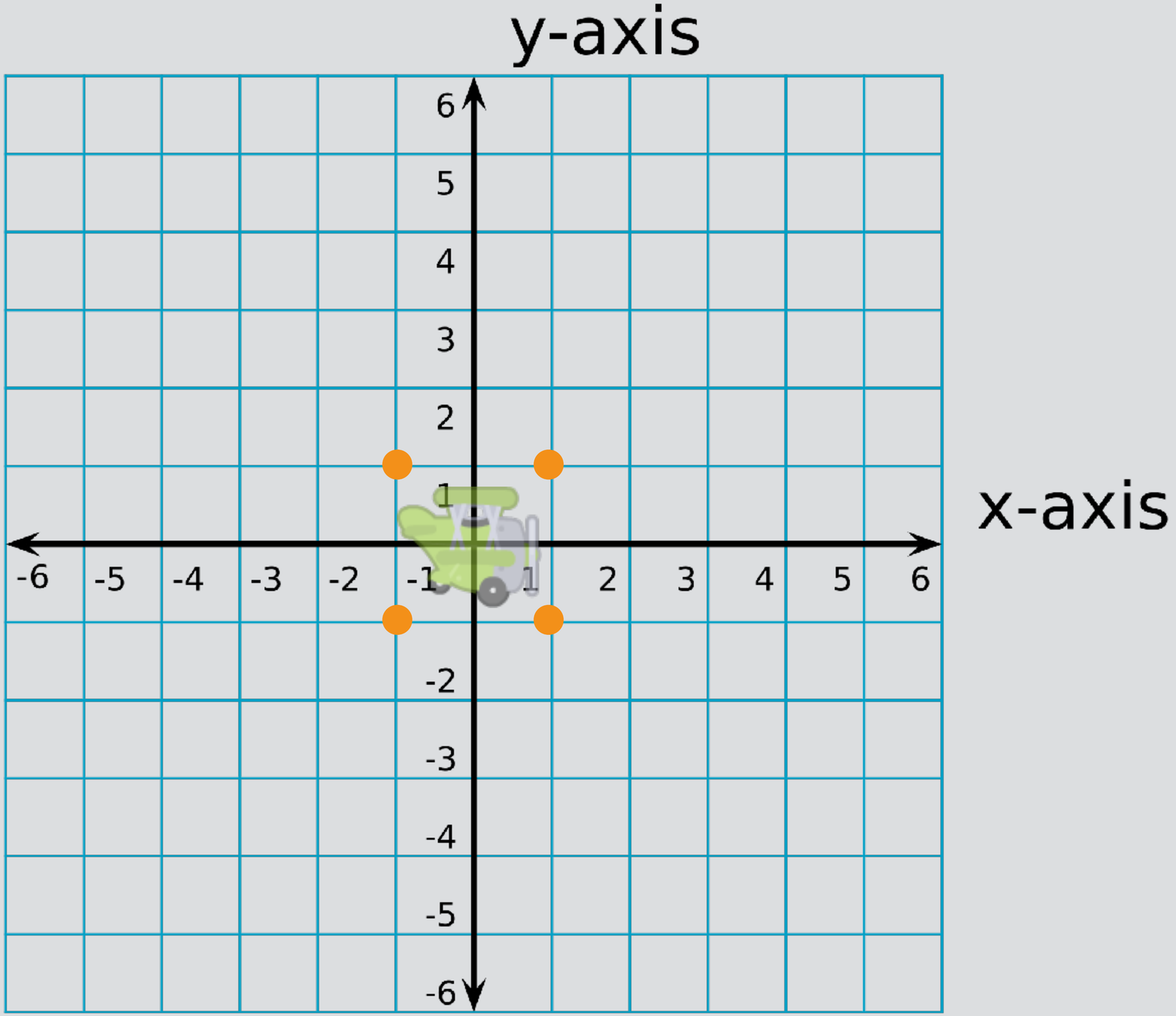

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

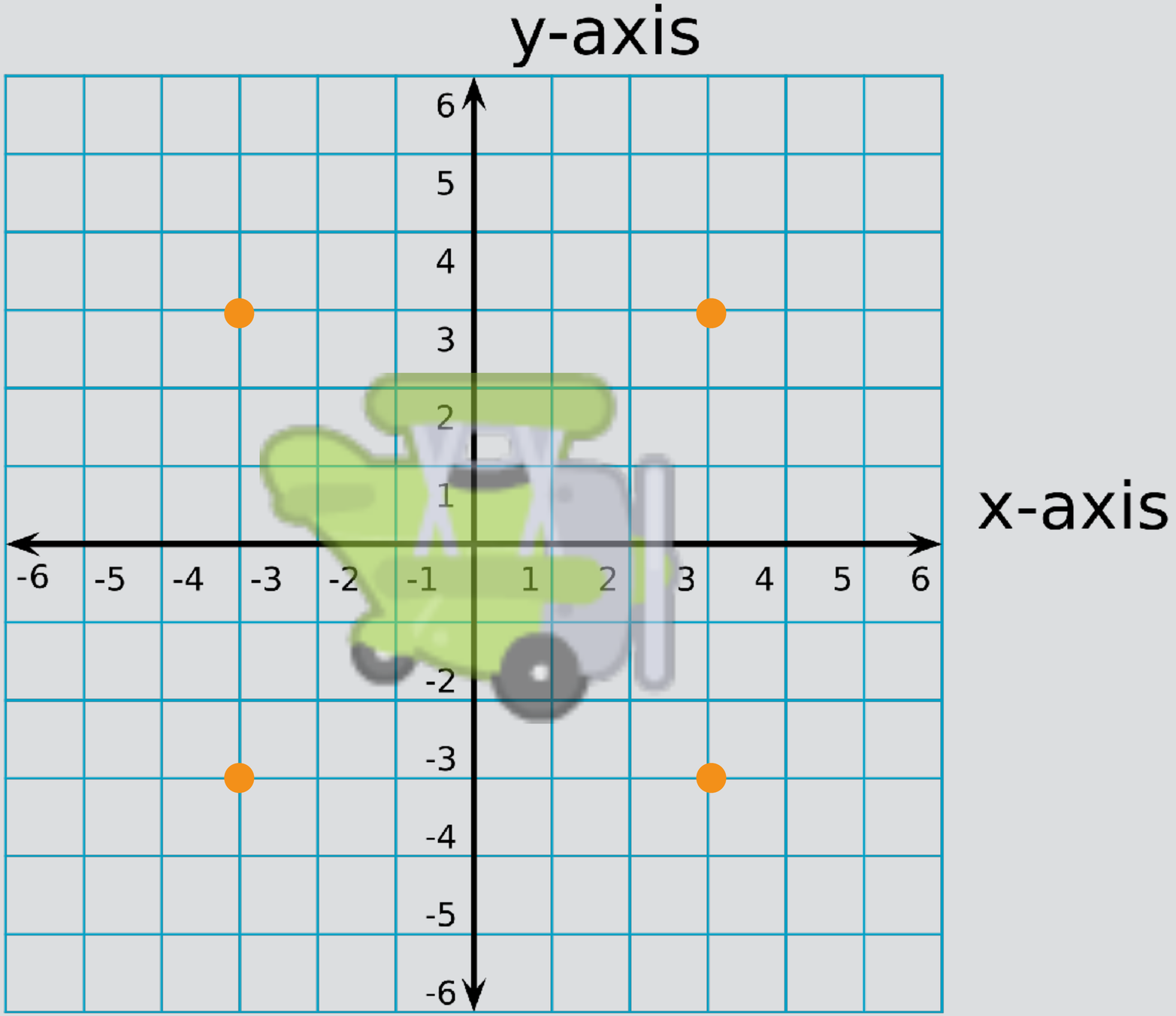
```
glm::mat4 glm::translate(glm::mat4 matrix, glm::vec3 translation);
```

Translates a matrix by specified coordinates.

```
modelMatrix = glm::mat4(1.0f);
modelMatrix = glm::translate(modelMatrix, glm::vec3(2.0f, 0.0f, 1.0f));
```

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

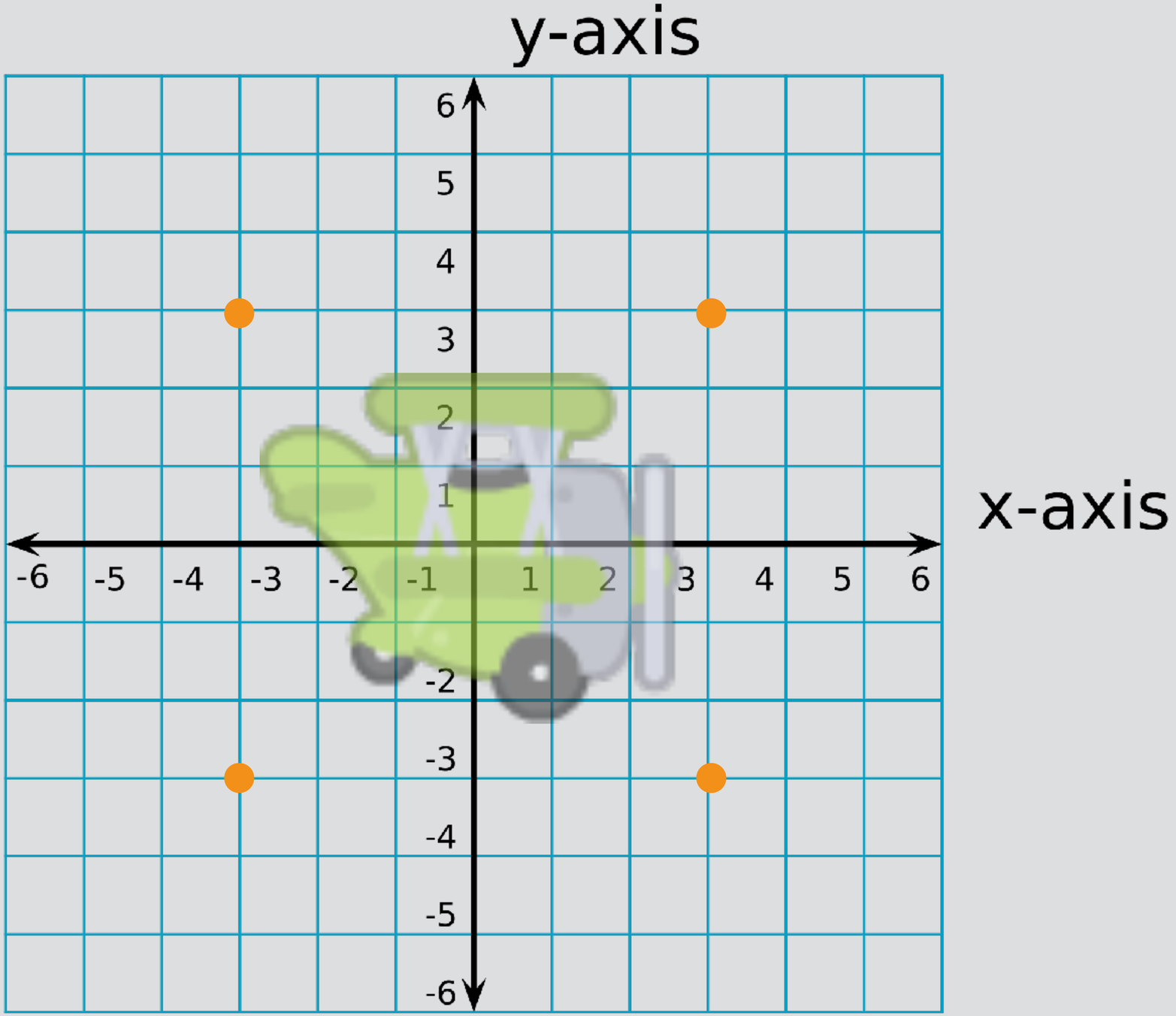

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

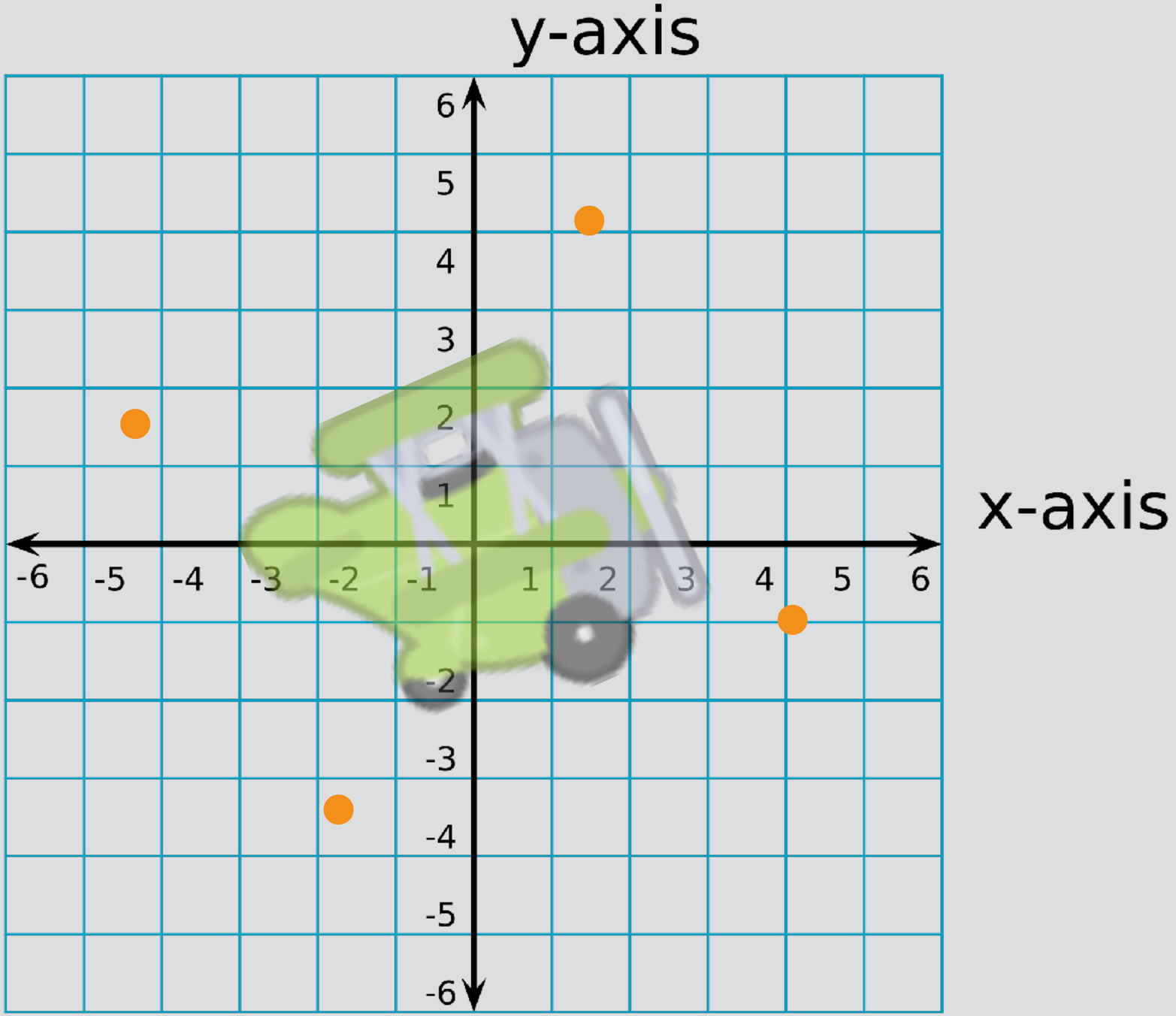
```
glm::mat4 glm::scale(glm::mat4 matrix, glm::vec3 scale);
```

Scales a matrix by specified coordinates.

```
modelMatrix = glm::mat4(1.0f);
modelMatrix = glm::scale(modelMatrix, glm::vec3(2.0f, 2.0f, 1.0f));
```

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

```
glm::mat4 glm::rotate(glm::mat4 matrix, float angle, glm::vec3 axis);
```

Rotates a matrix by specified radians.

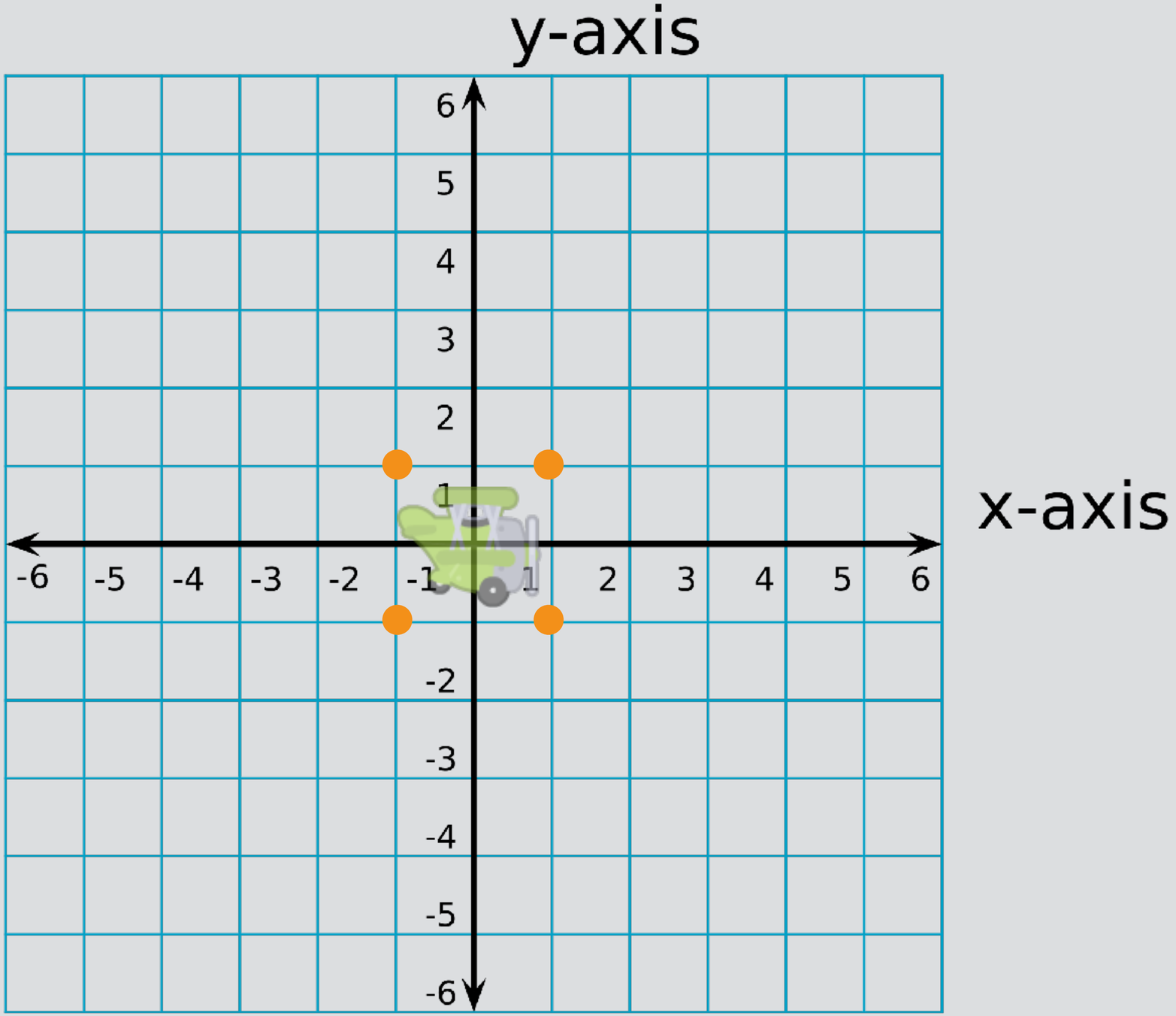
```
modelMatrix = glm::mat4(1.0f);
float angle = 45.0f * (3.1415926f / 180.0f);
modelMatrix = glm::rotate(modelMatrix, angle, glm::vec3(0.0f, 0.0f, 1.0f));
```

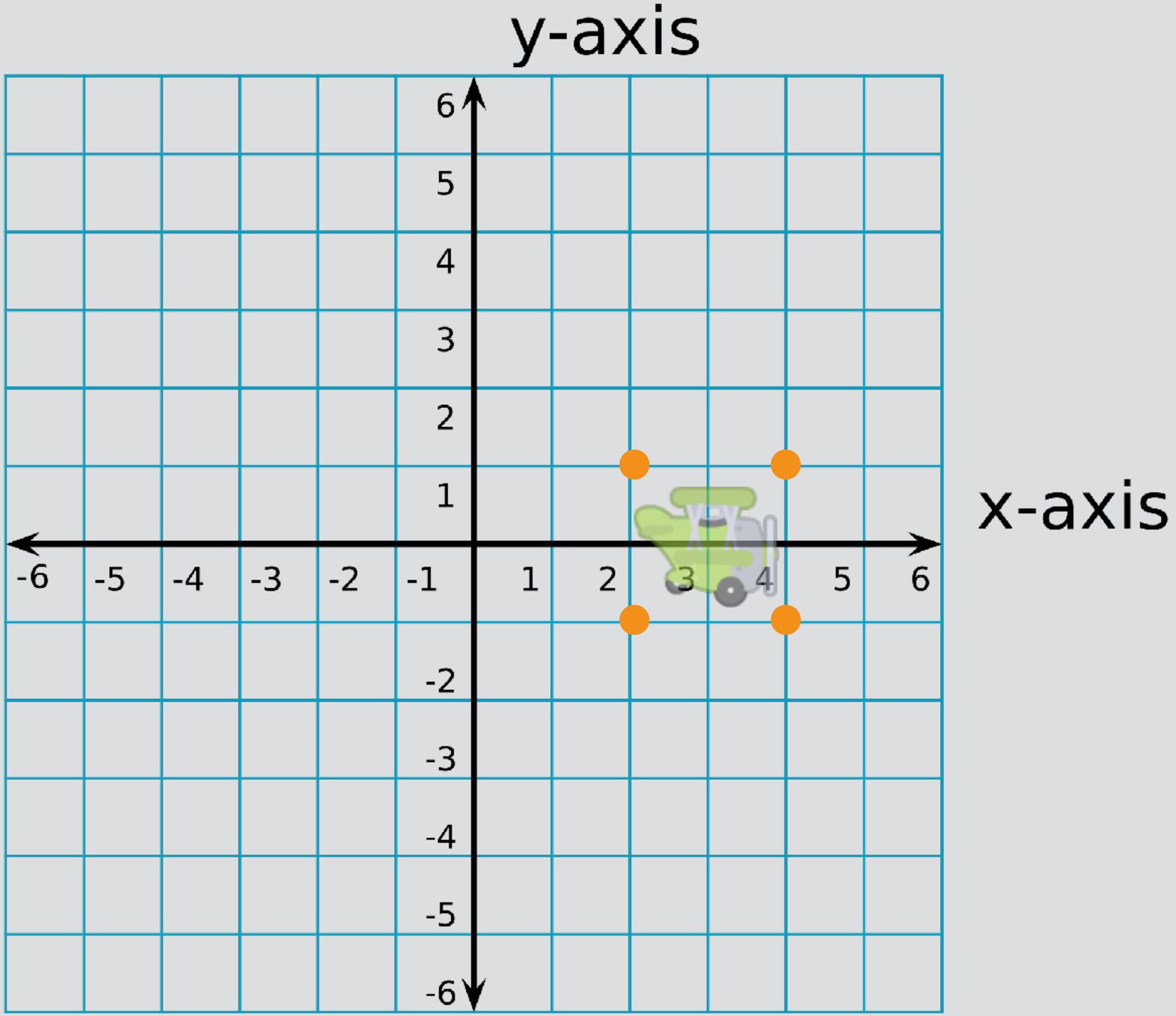
To convert from **degrees** to **radians**:

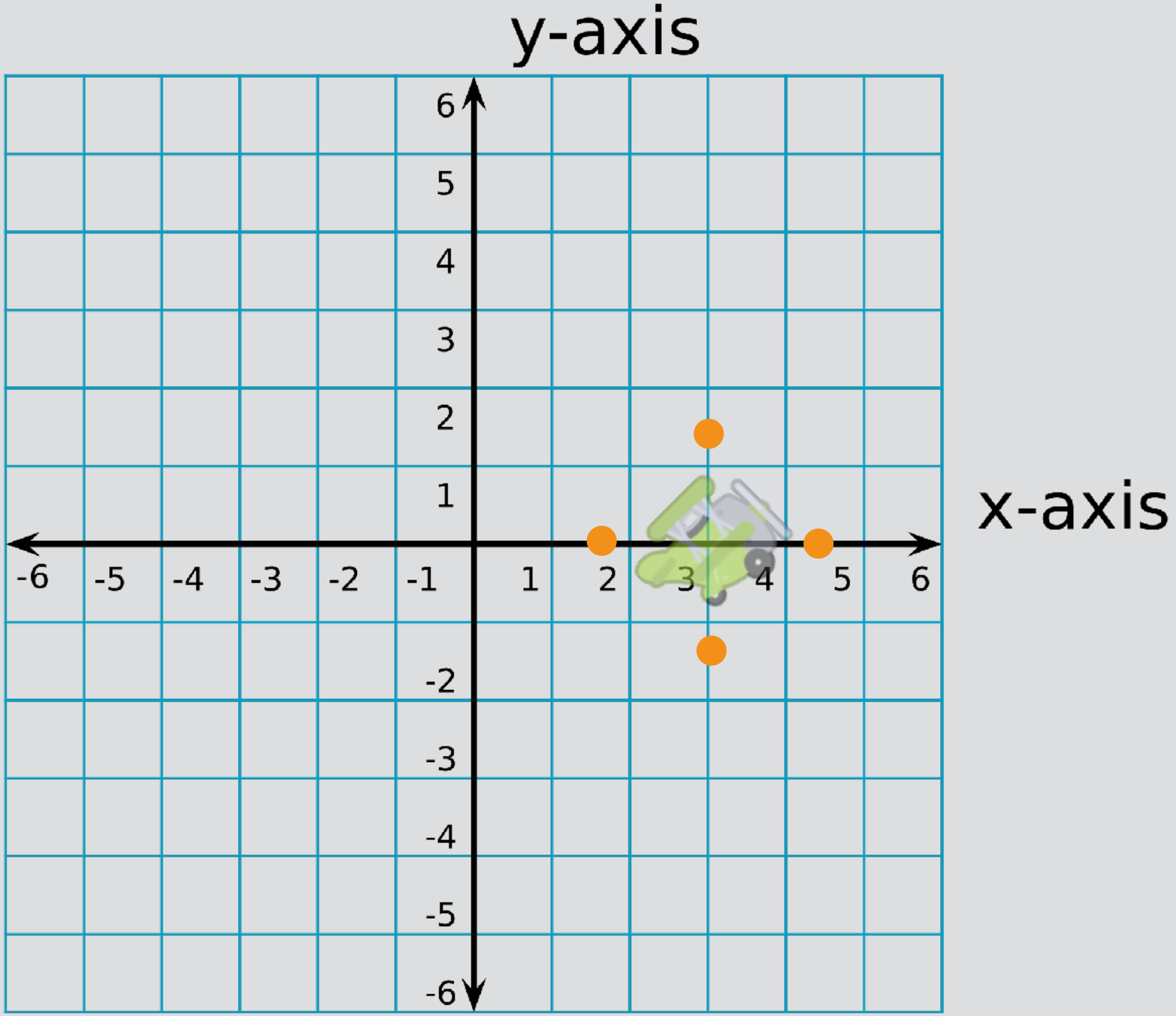
RADIANS = DEGREES * (PI / 180)

Matrix transformations
are not commutative!

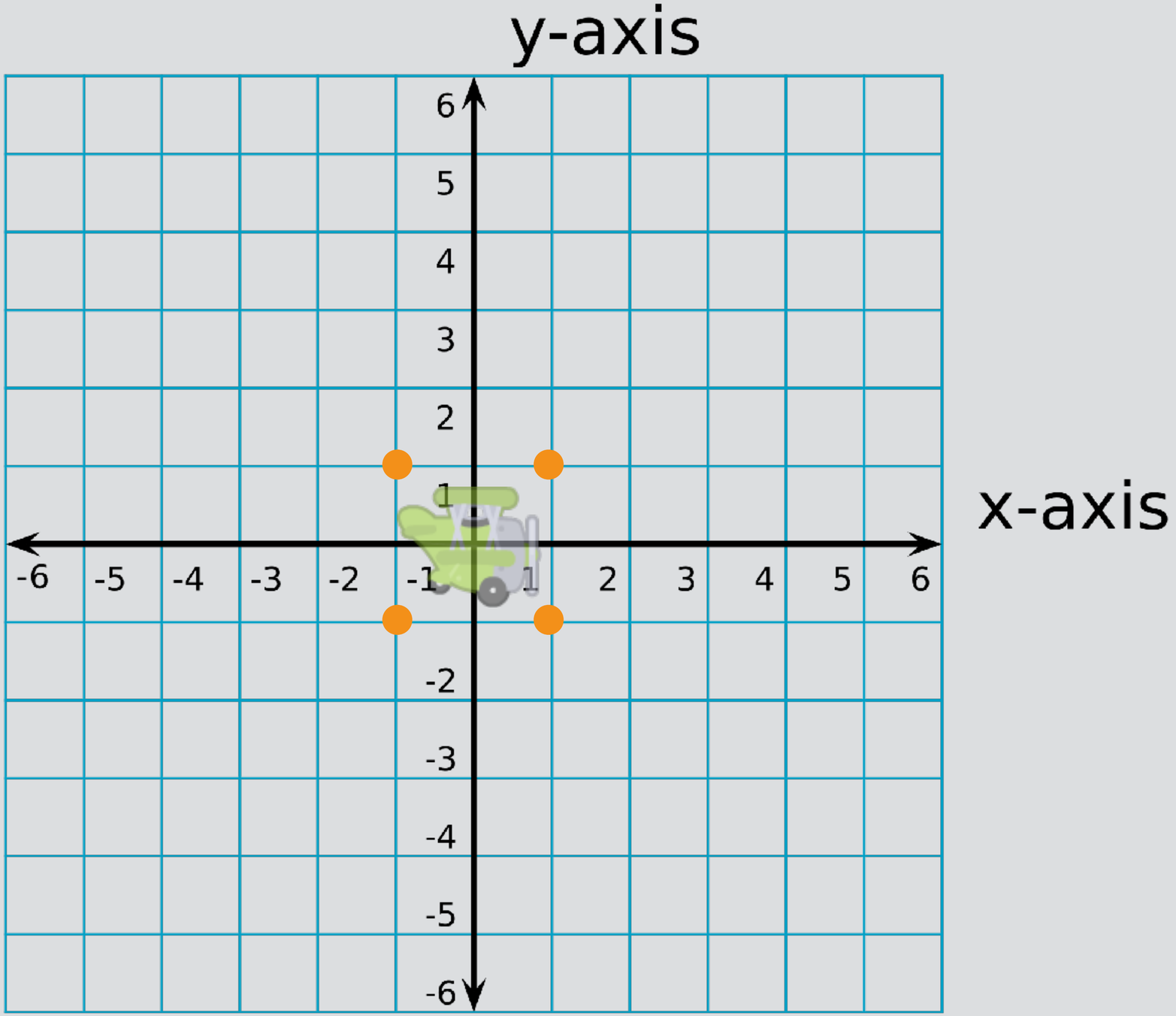
```
glm::mat4 modelMatrix = glm::mat4(1.0f);

modelMatrix = glm::translate(modelMatrix, glm::vec3(3.0f, 0.0f, 0.0f));

float angle = 45.0f * (3.1415926f / 180.0f);
modelMatrix = glm::rotate(modelMatrix, angle, glm::vec3(0.0f, 0.0f, 1.0f));
```

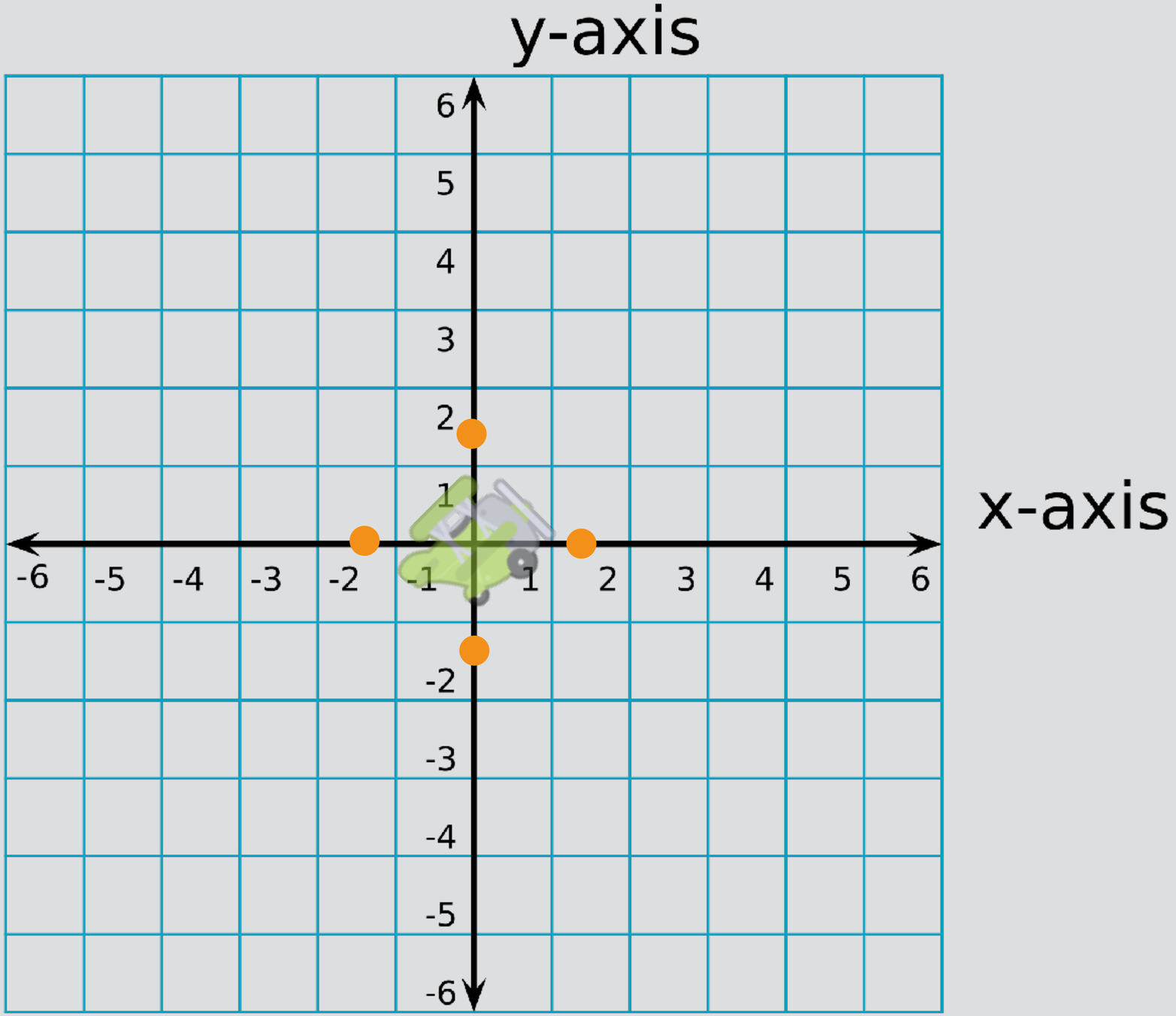

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

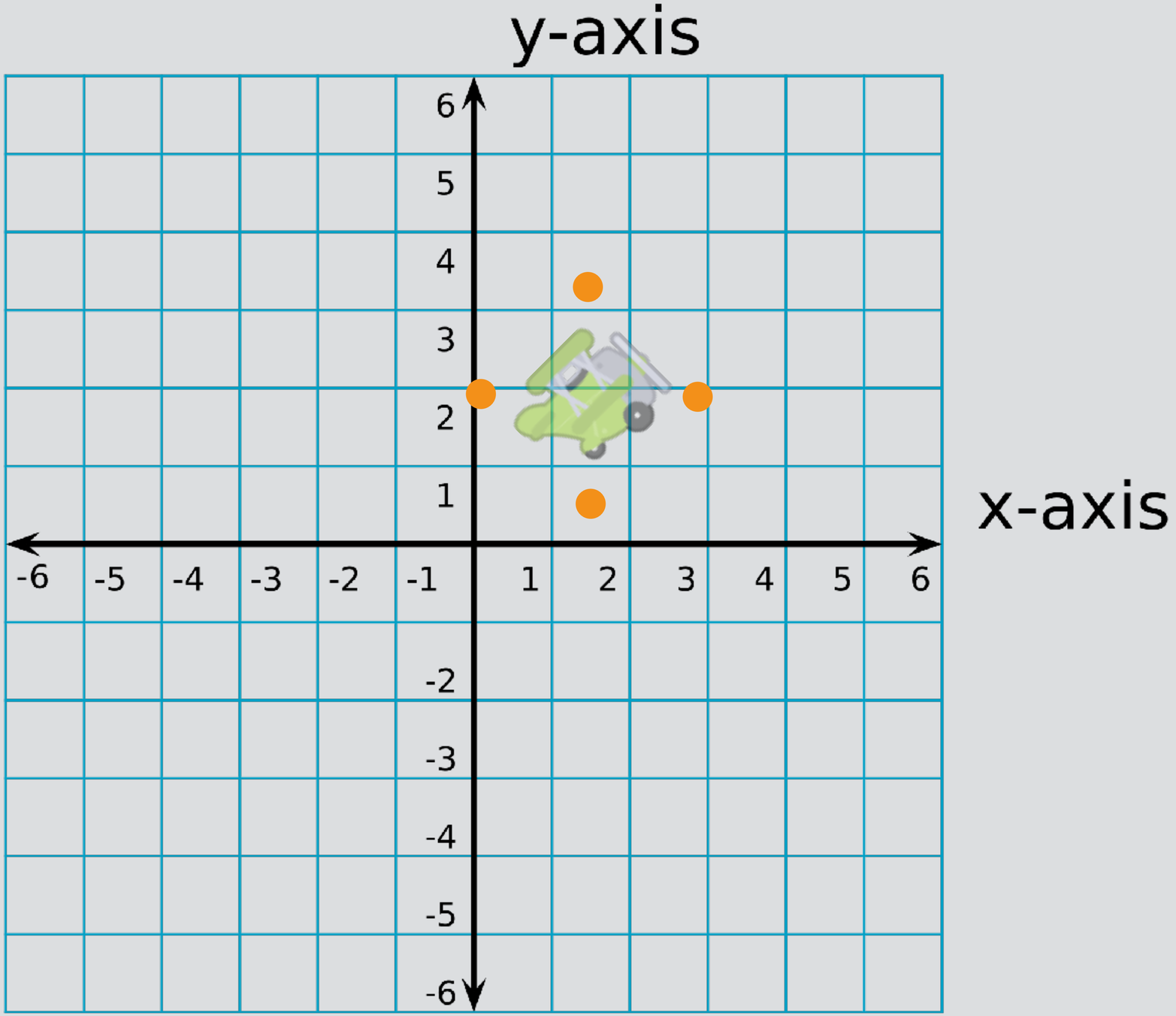

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6


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

```
glm::mat4 modelMatrix = glm::mat4(1.0f);

float angle = 45.0f * (3.1415926f / 180.0f);
modelMatrix = glm::rotate(modelMatrix, angle, glm::vec3(0.0f, 0.0f, 1.0f));

modelMatrix = glm::translate(modelMatrix, glm::vec3(3.0f, 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

Multiple transformations.

```
glm::mat4 modelMatrix = glm::mat4(1.0f);
modelMatrix = glm::translate(modelMatrix, glm::vec3(1.0f, 0.0f, 0.0f));

program.SetModelMatrix(modelMatrix);

// draw first object

modelMatrix = glm::mat4(1.0f);
modelMatrix = glm::translate(modelMatrix, glm::vec3(-2.0f, 0.0f, 0.0f));

program.SetModelMatrix(modelMatrix);

// draw second object
```

```
glm::mat4 modelMatrix1 = glm::mat4(1.0f);
modelMatrix1 = glm::translate(modelMatrix1, glm::vec3(1.0f, 0.0f, 0.0f));
program.SetModelMatrix(modelMatrix1);

// draw first object

glm::mat4 modelMatrix2 = glm::mat4(1.0f);
modelMatrix2 = glm::translate(modelMatrix2, glm::vec3(-2.0f, 0.0f, 0.0f));
program.SetModelMatrix(modelMatrix2);

// draw second object
```