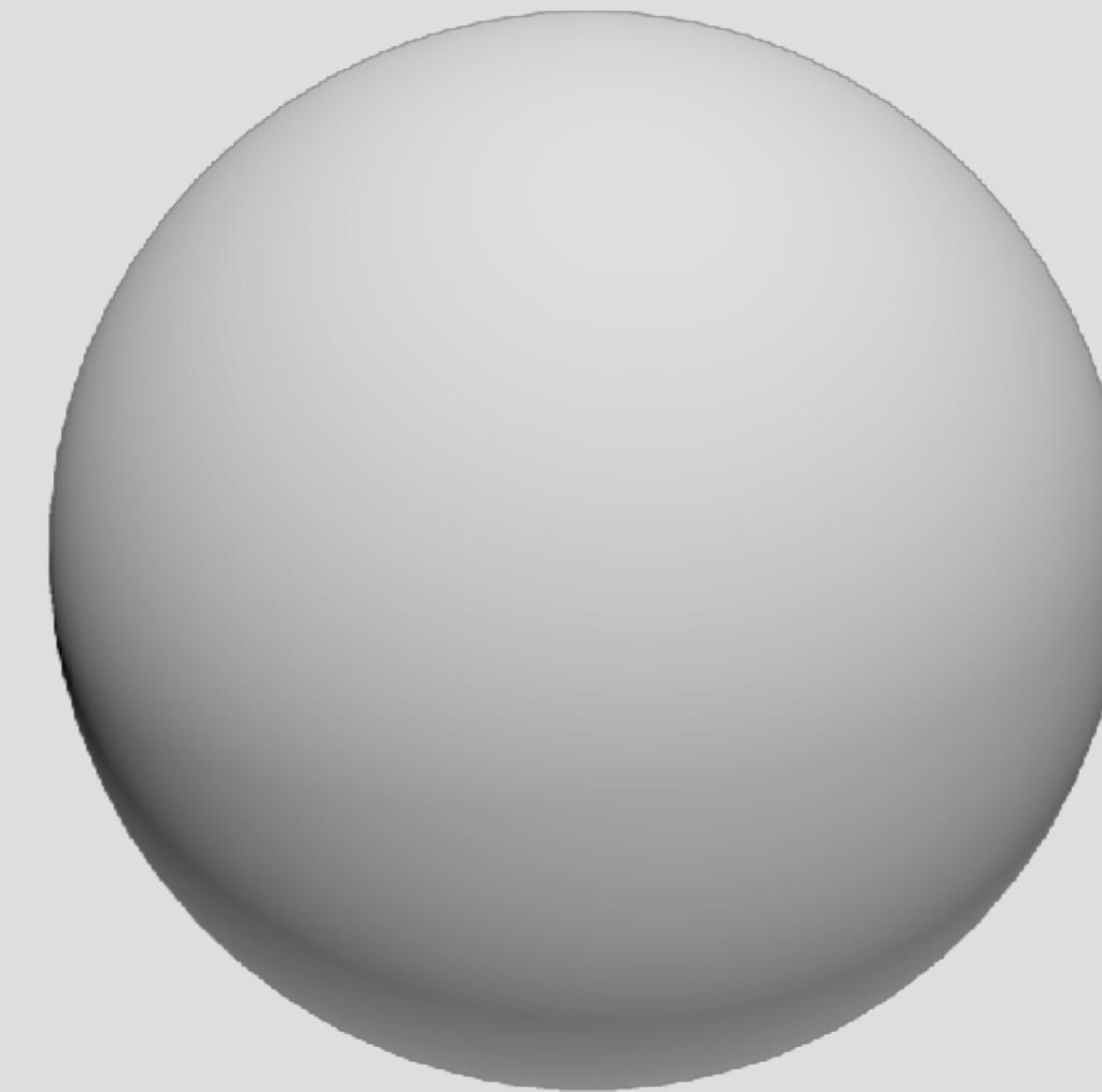


Graphics Foundations Part 1



CS 3113

Computer graphics

Defend

Put Away

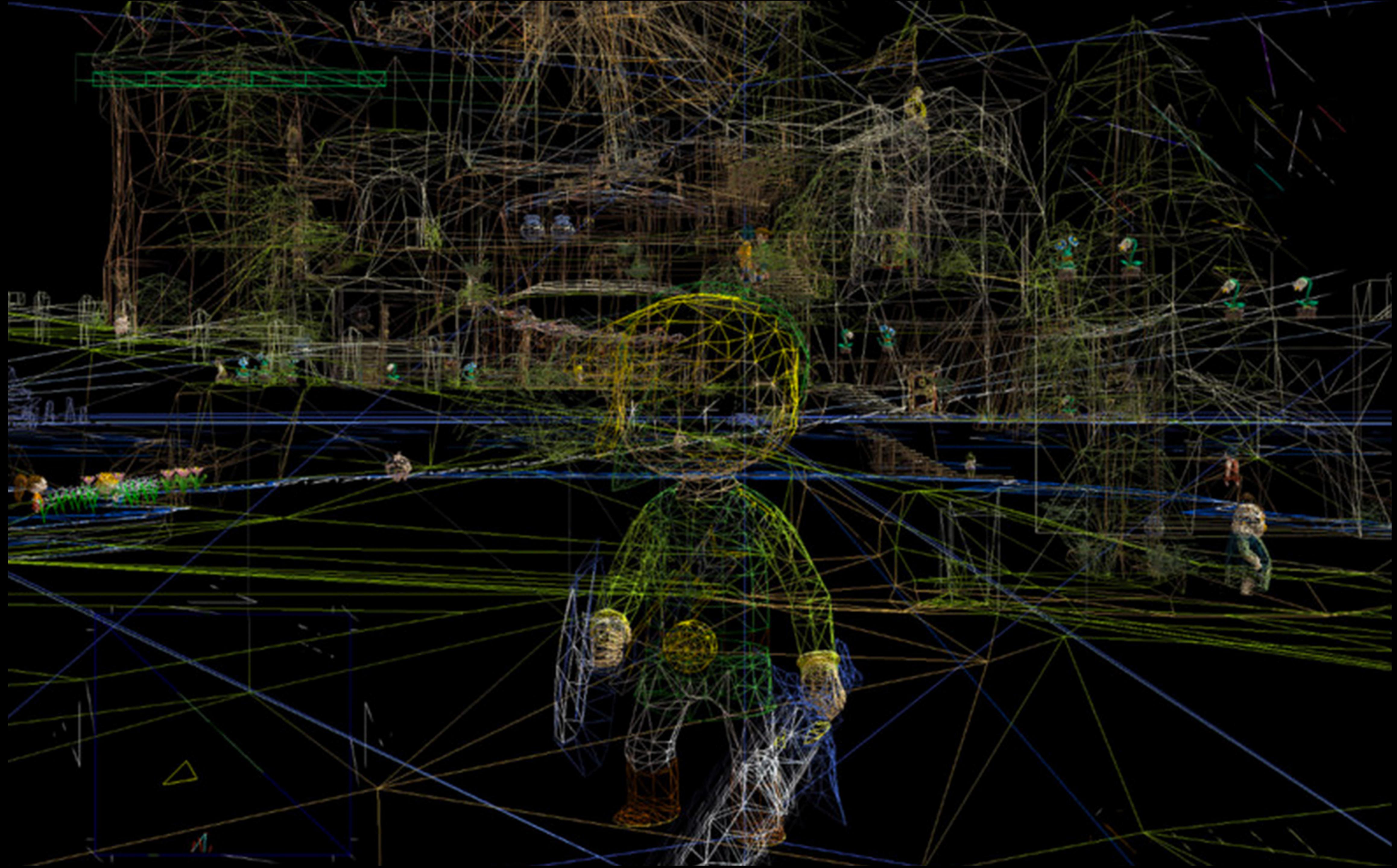
76

72

FREE

4168

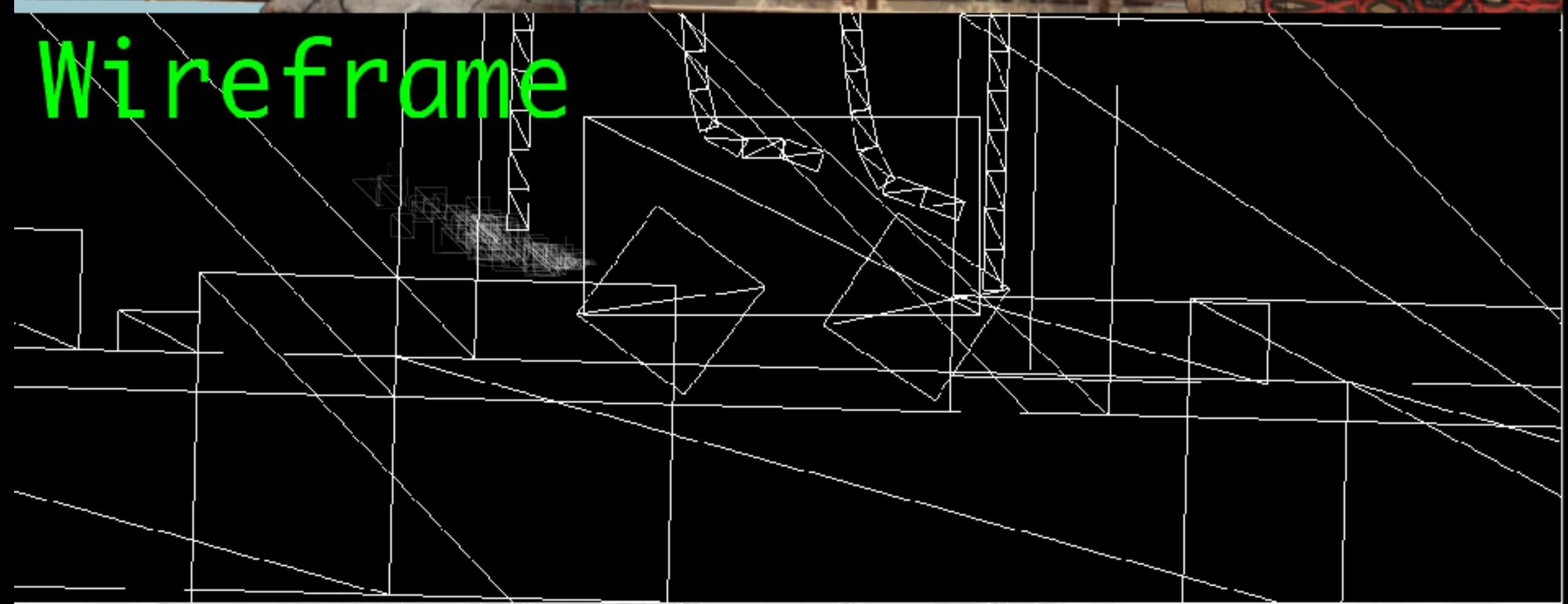


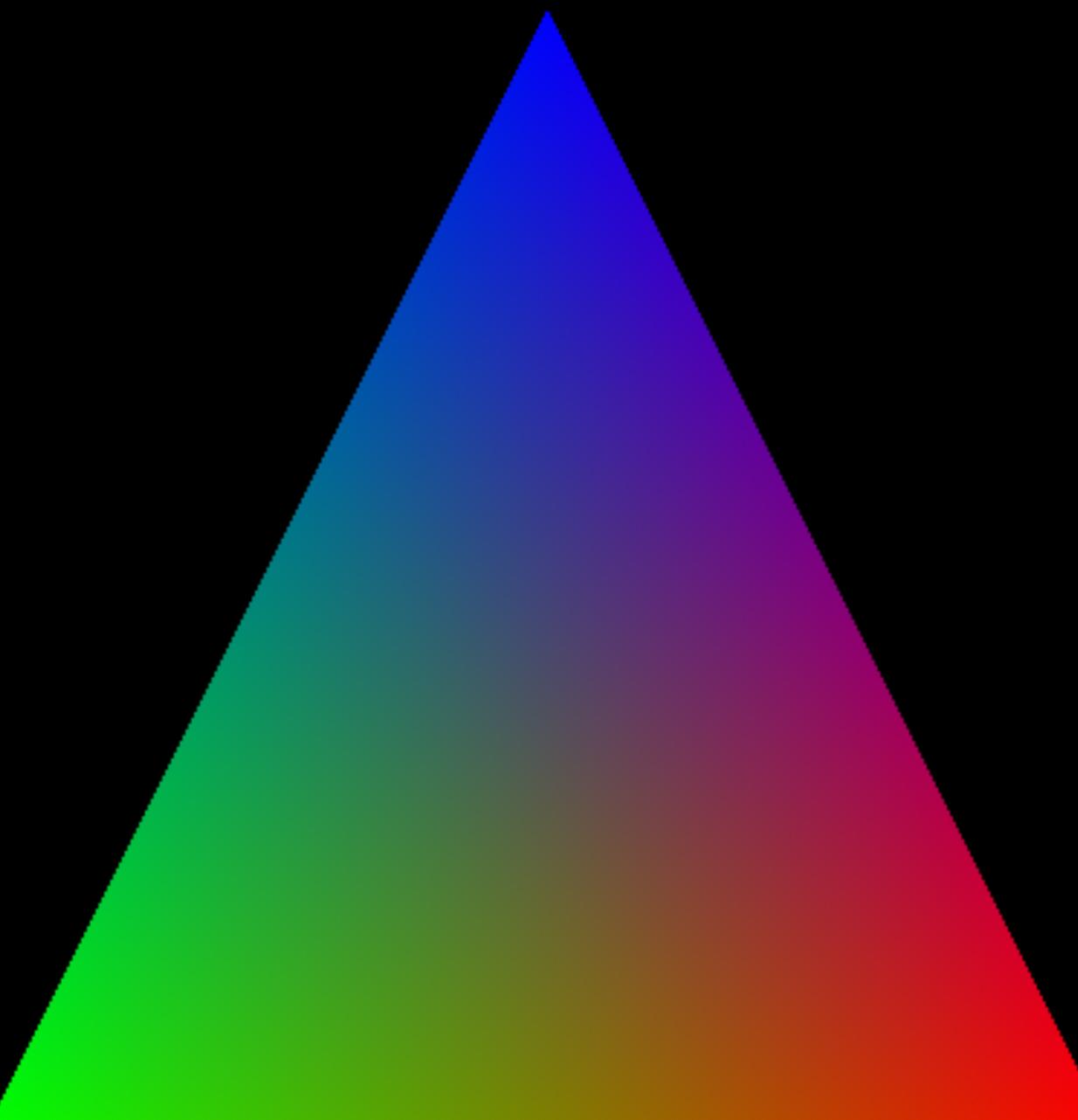












Digital images

Image on the screen

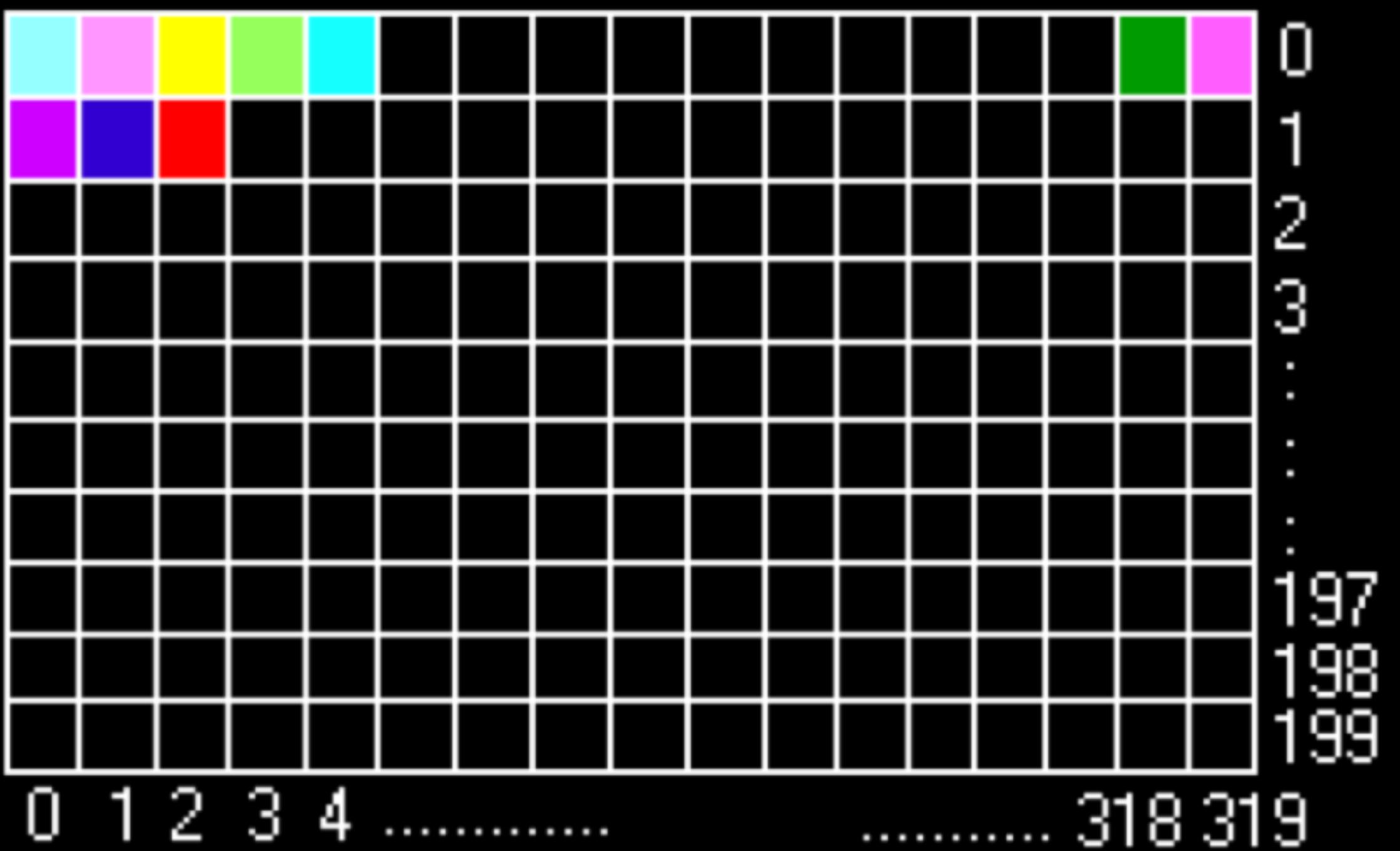


Image in memory

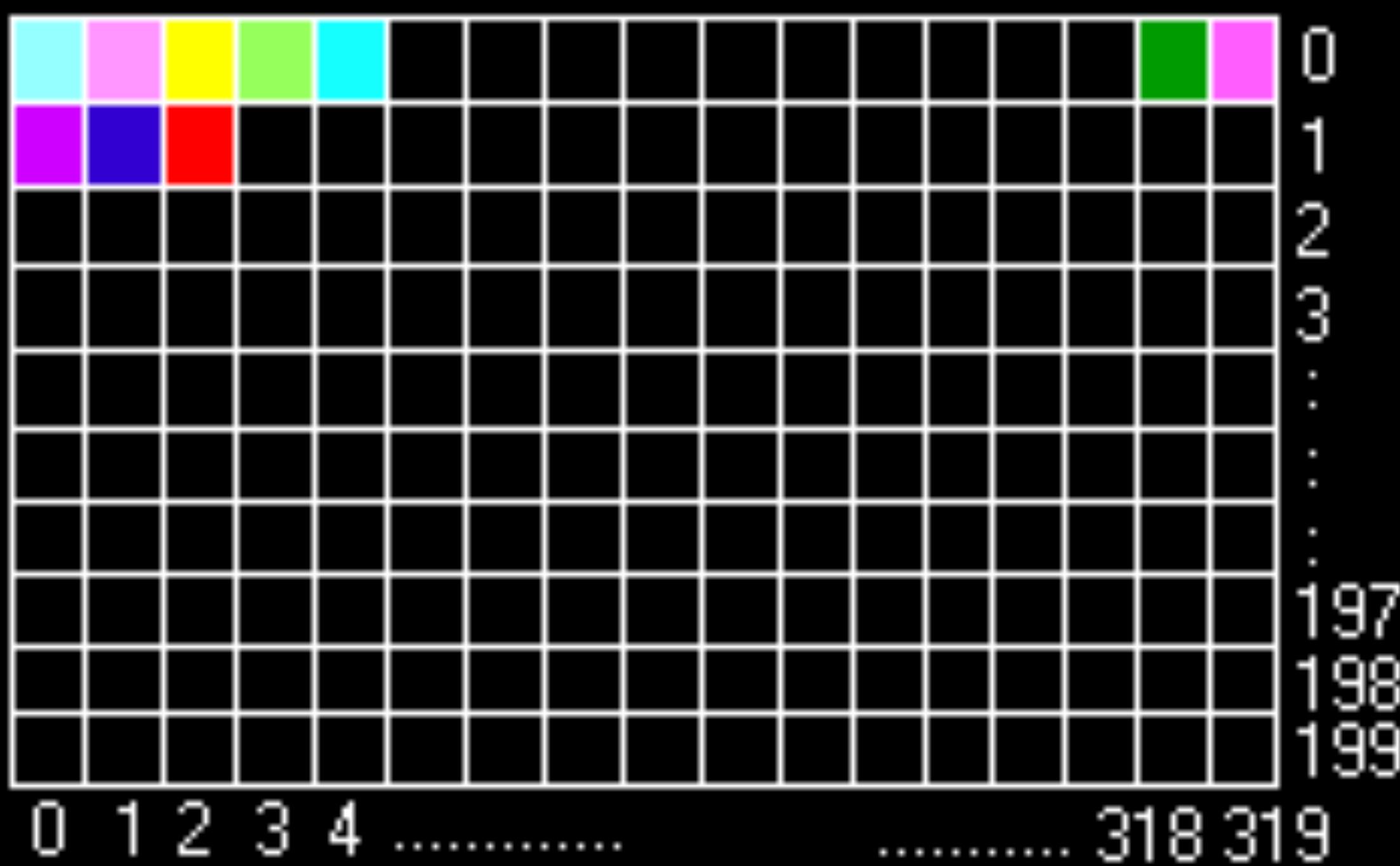


Graphics in games



Software rendering

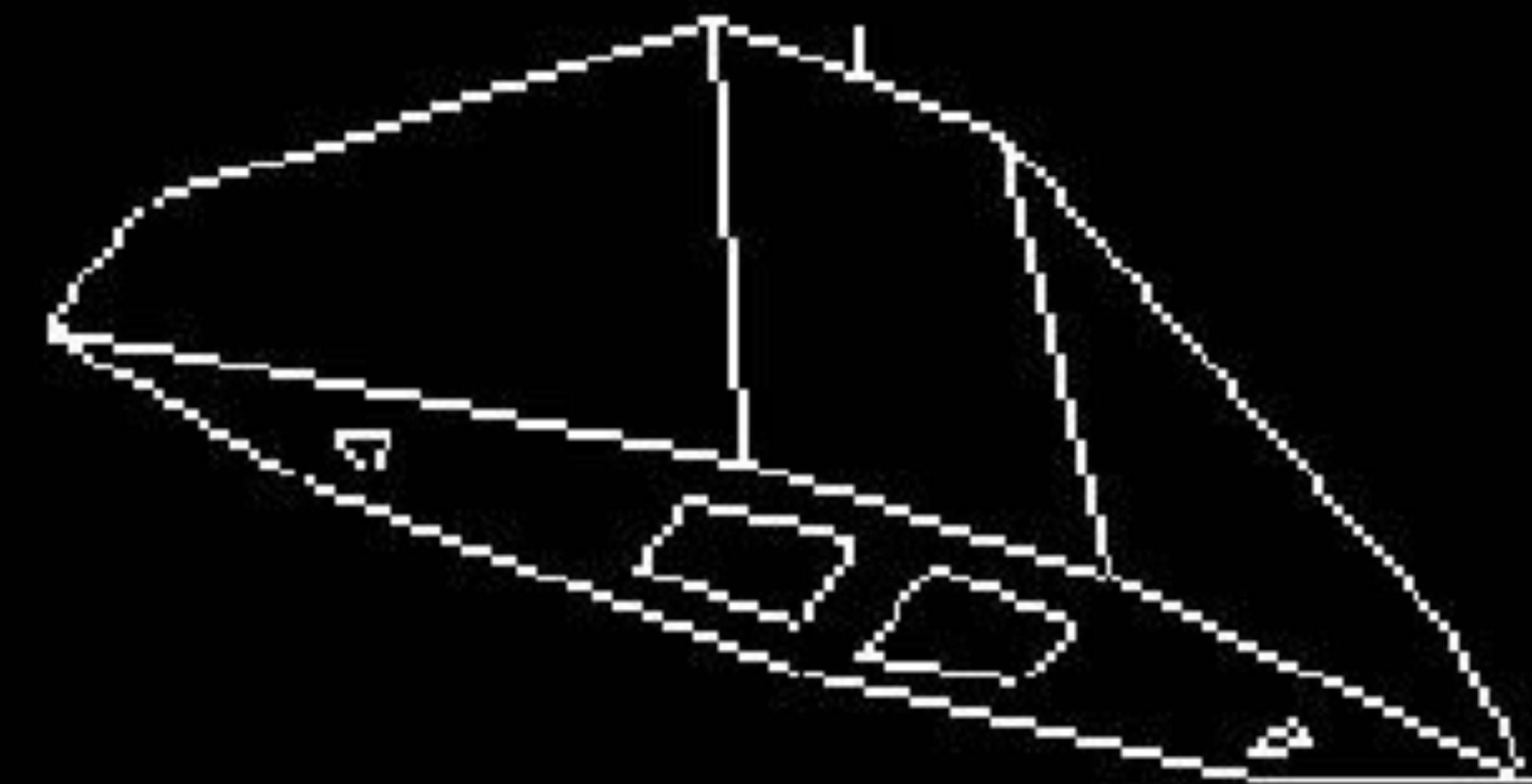
The screen



The video memory



Software rendering



Load New Commander (Y/N)?

SPD: 0
USI: 01.78
THR:90%

HDG:0

ALT:83







CHAR

QUESTS

MAP

MENU



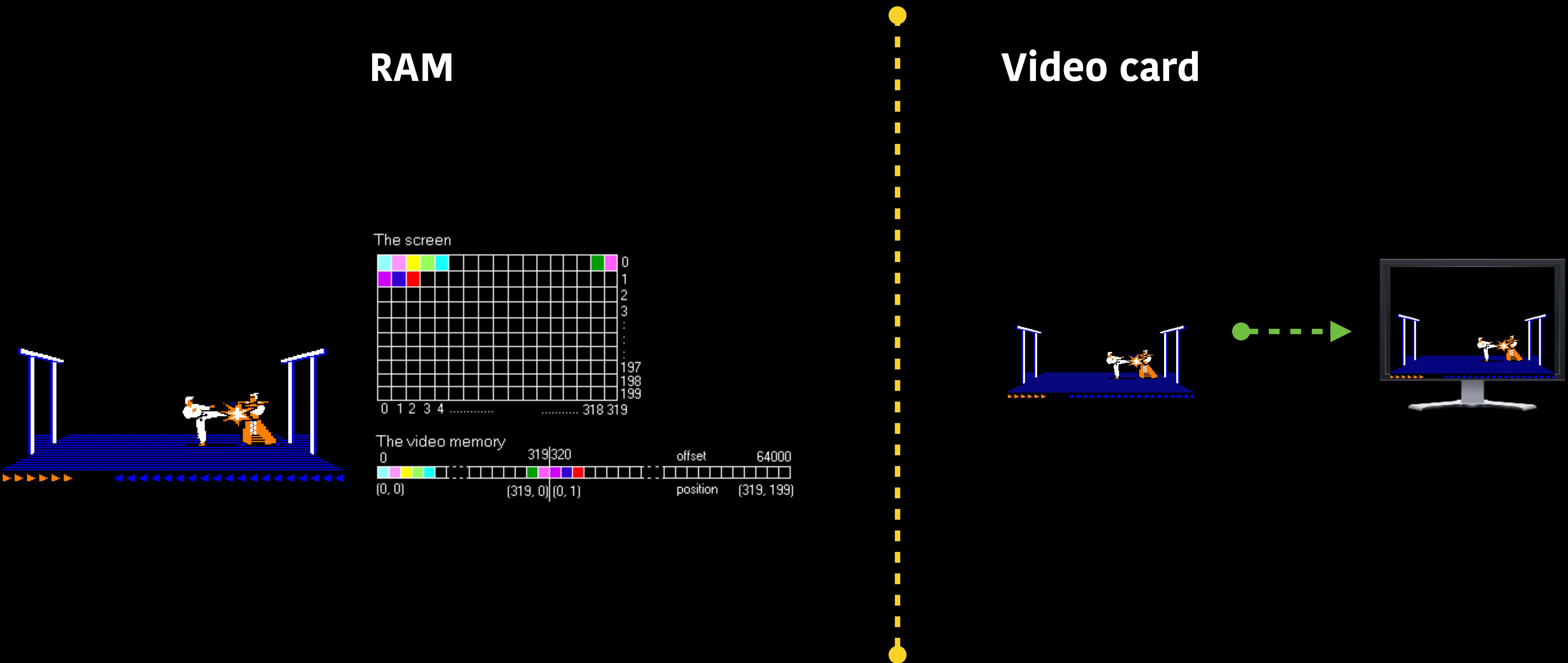
RED STORM
TOTAL KILLS : 19
RESISTS : LIGHTNING
IMMUNE : MAGIC

INV

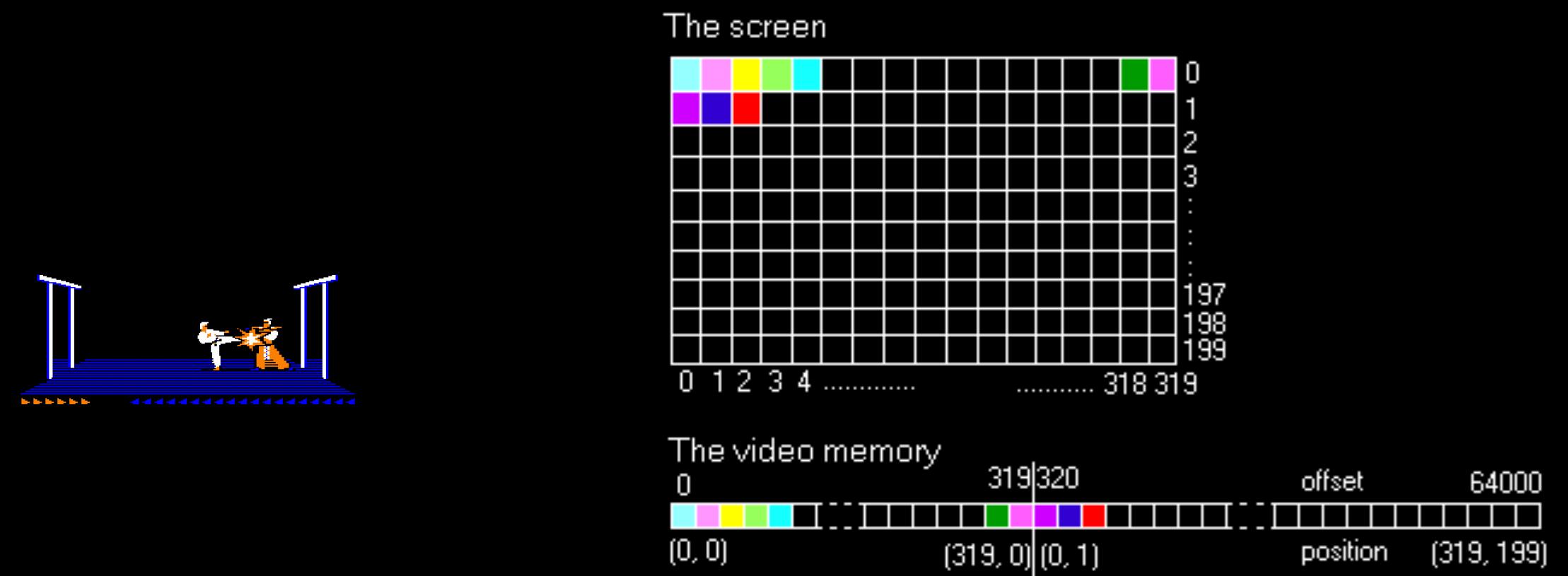
SPELLS



Software Rendering



Software Rendering

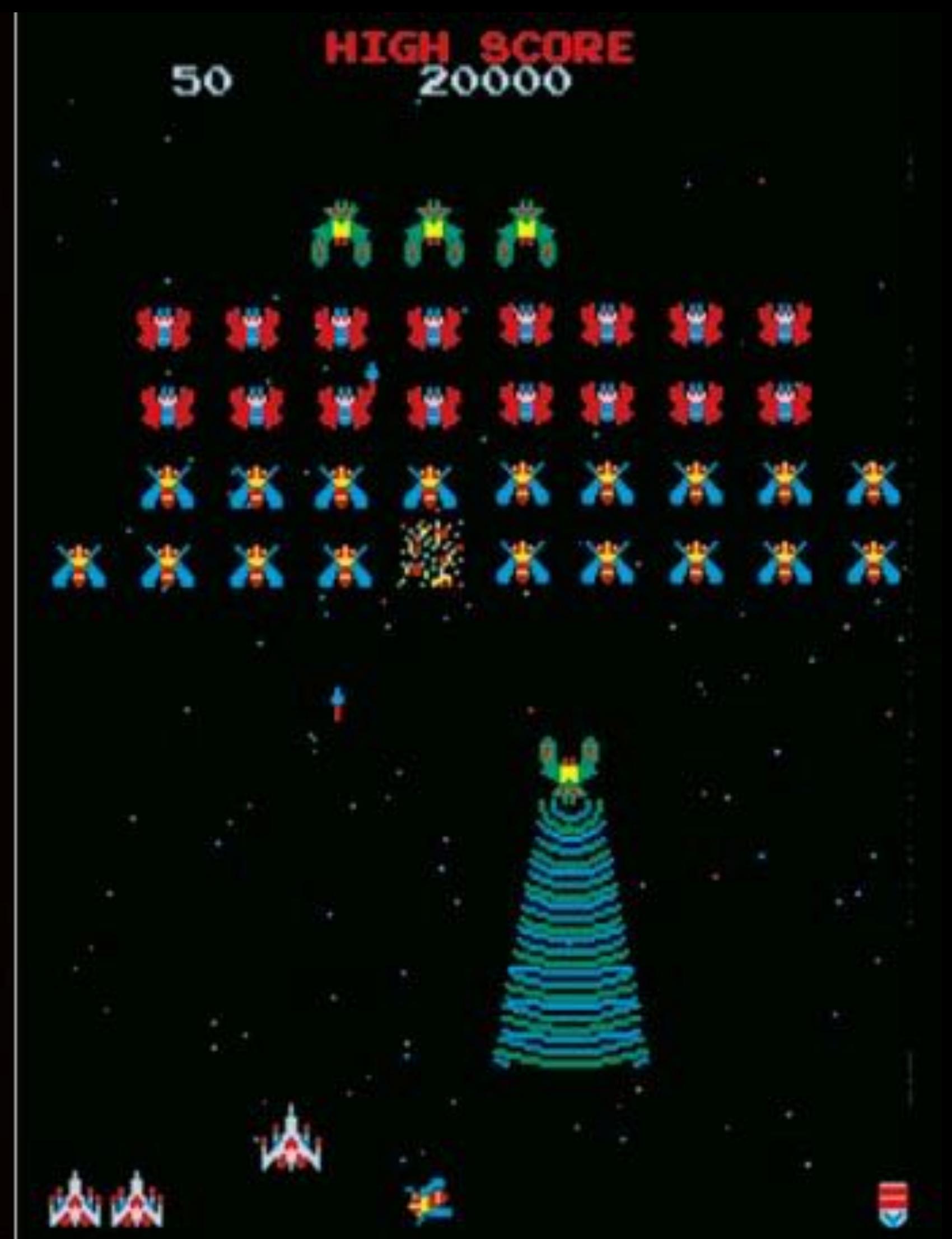


2560 * 1600 = 4,096,000 pixels

= ~12 MB of data!

Hardware rendering







MARIO
000000

0 x 00

WORLD TIME
1-1

SUPER MARIO BROS.

@1985 NINTENDO

• 1 PLAYER GAME

2 PLAYER GAME

TOP - 000000



offset = 2305



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

offset = 2049



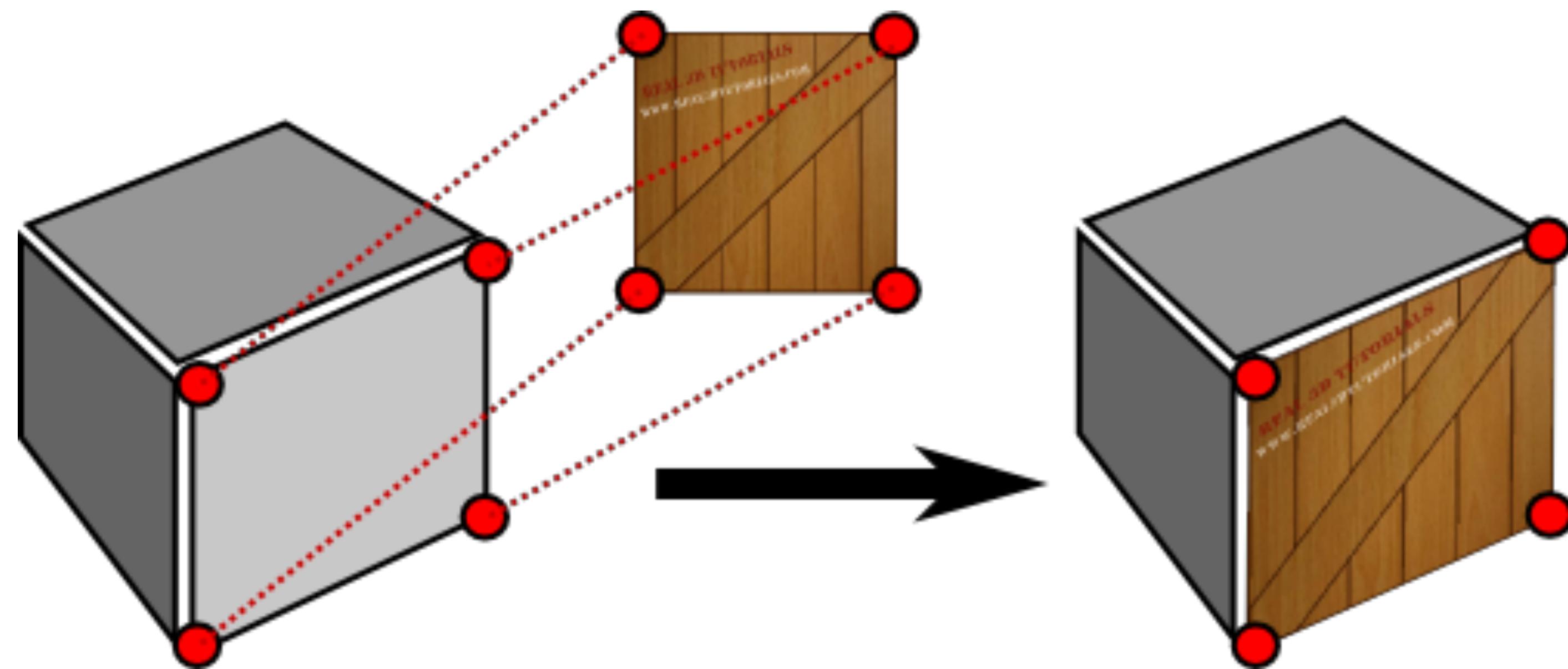
\$23C0	\$23C1	\$23C2	\$23C3	\$23C4	\$23C5	\$23C6	\$23C7
000000	000000	000000	000000	000000	000000	000000	000000
\$23C8	\$23C9	\$23CA	\$23CB	\$23CC	\$23CD	\$23CE	\$23CF
\$23D0	\$23D1	\$23D2	\$23D3	\$23D4	\$23D5	\$23D6	\$23D7
\$23D8	\$23D9	\$23DA	\$23DB	\$23DC	\$23DD	\$23DE	\$23DF
\$23E0	\$23E1	\$23E2	\$23E3	\$23E4	\$23E5	\$23E6	\$23E7
\$23F8	\$23E9	\$23EA	\$23EB	\$23EC	\$23ED	\$23EE	\$23EF
\$23F0	\$23F1	\$23F2	\$23F3	\$23F4	\$23F5	\$23F6	\$23F7
\$23F8	\$23F9	\$23FA	\$23FB	\$23FC	\$23FD	\$23FE	\$23FF

RACE LEADER

USA

RACE LEAD





12MB

BLASTER

Voodoo2

Hardware AWARD
PC-Gamer 05/98

Hardware AWARD
PC-Aktion 05/98

PC Award
PC-Player 05/98

4
FASZINIERENDE SPIELE

SP

INCOMING

actua SOCCER 2

ULTIMATE RACE PRO

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

CREATIVE
WWW.SOUNDBLASTER.COM





110



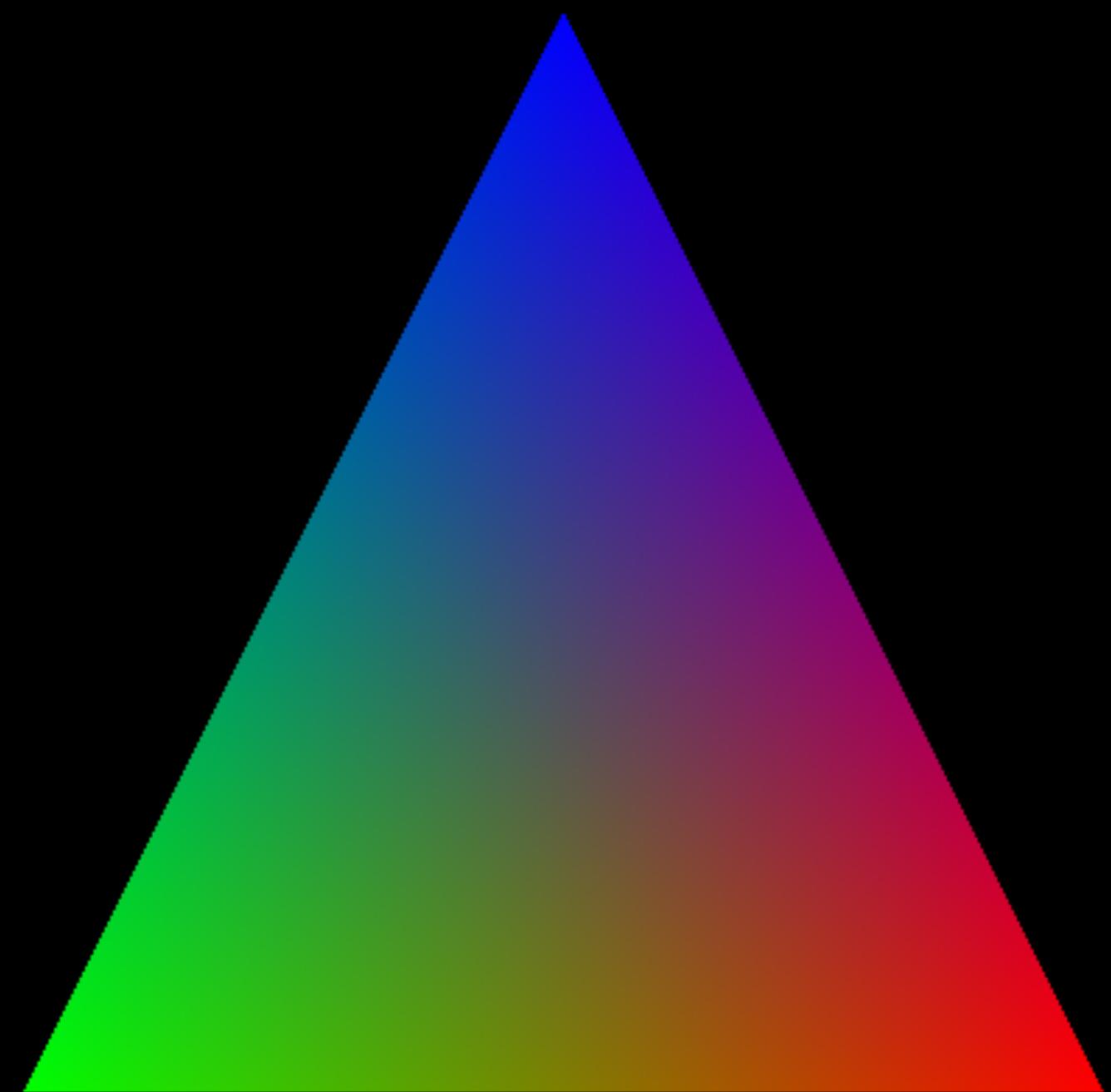
108



20 0 0





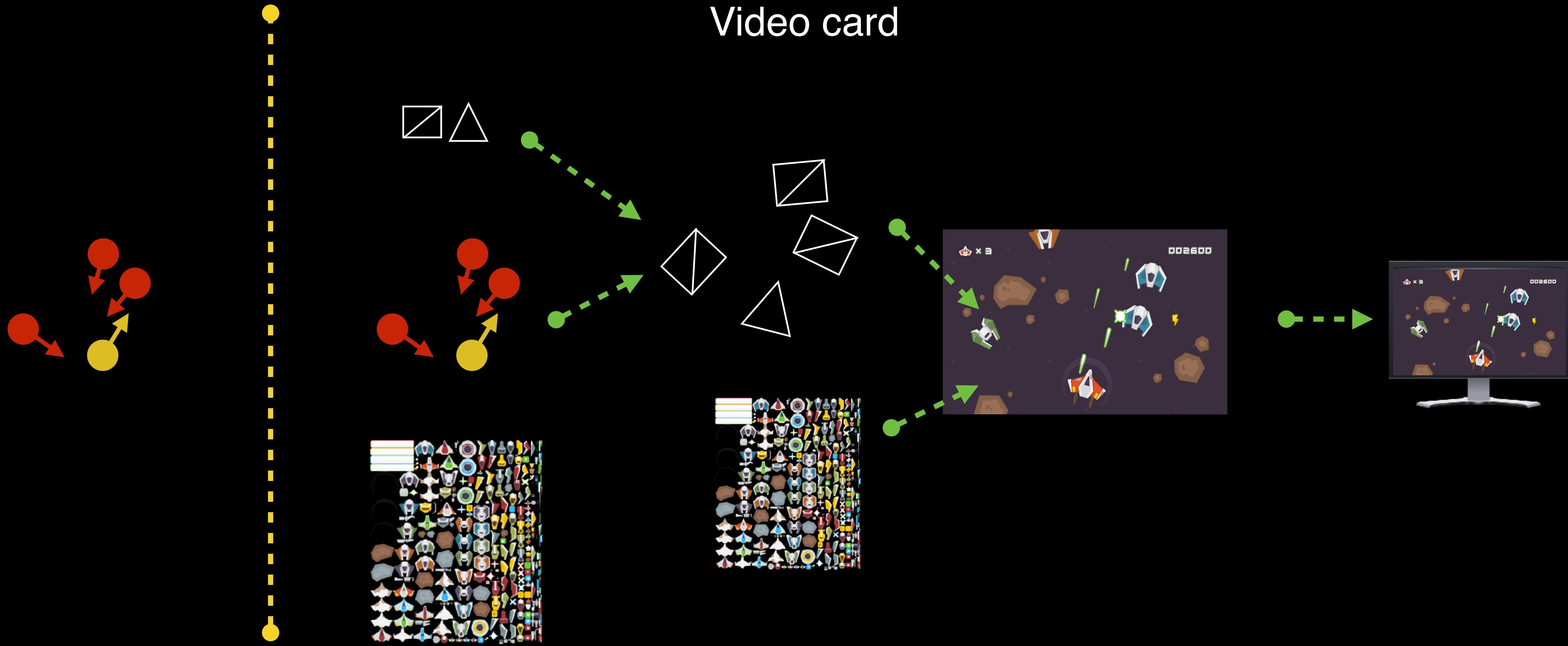


Hardware Rendering



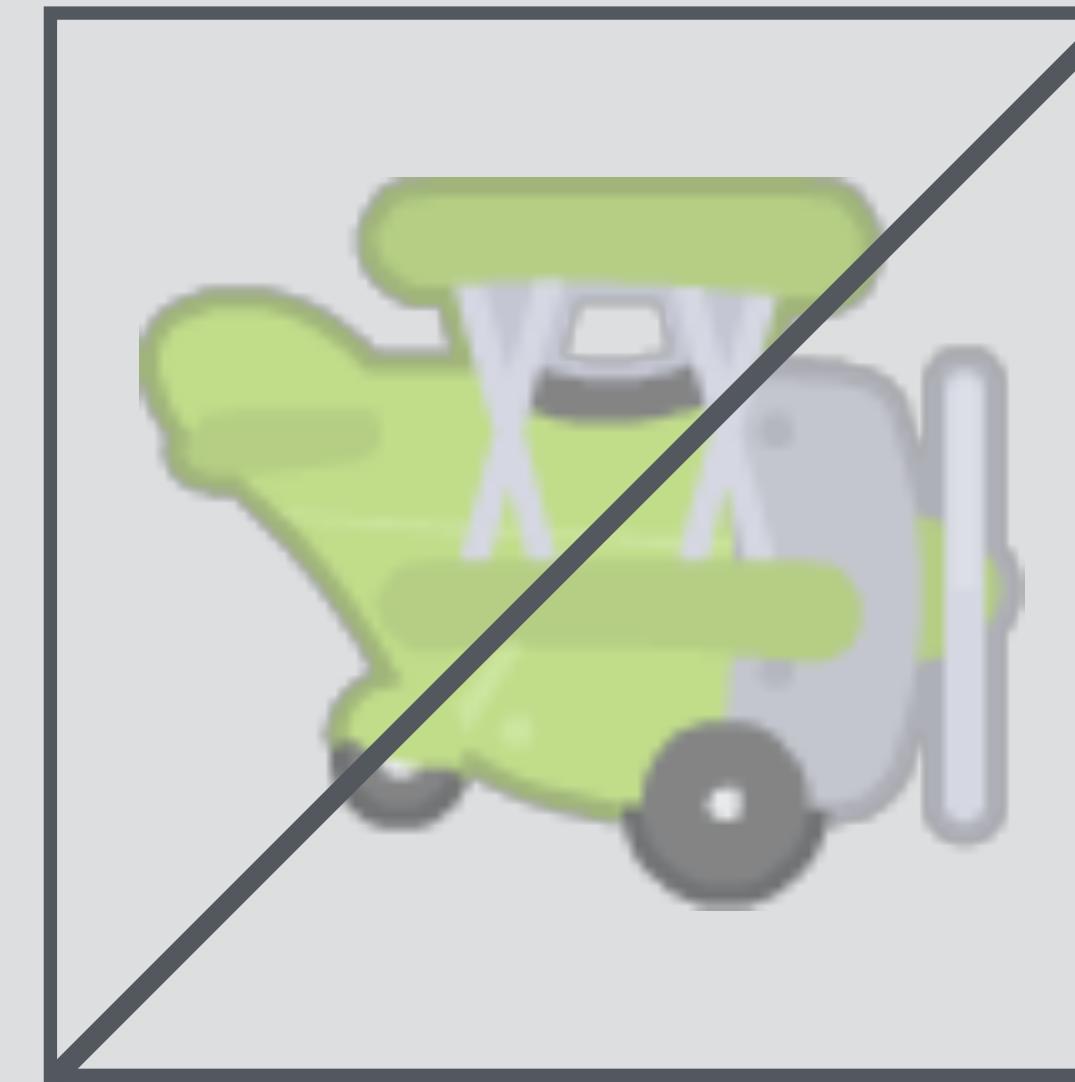
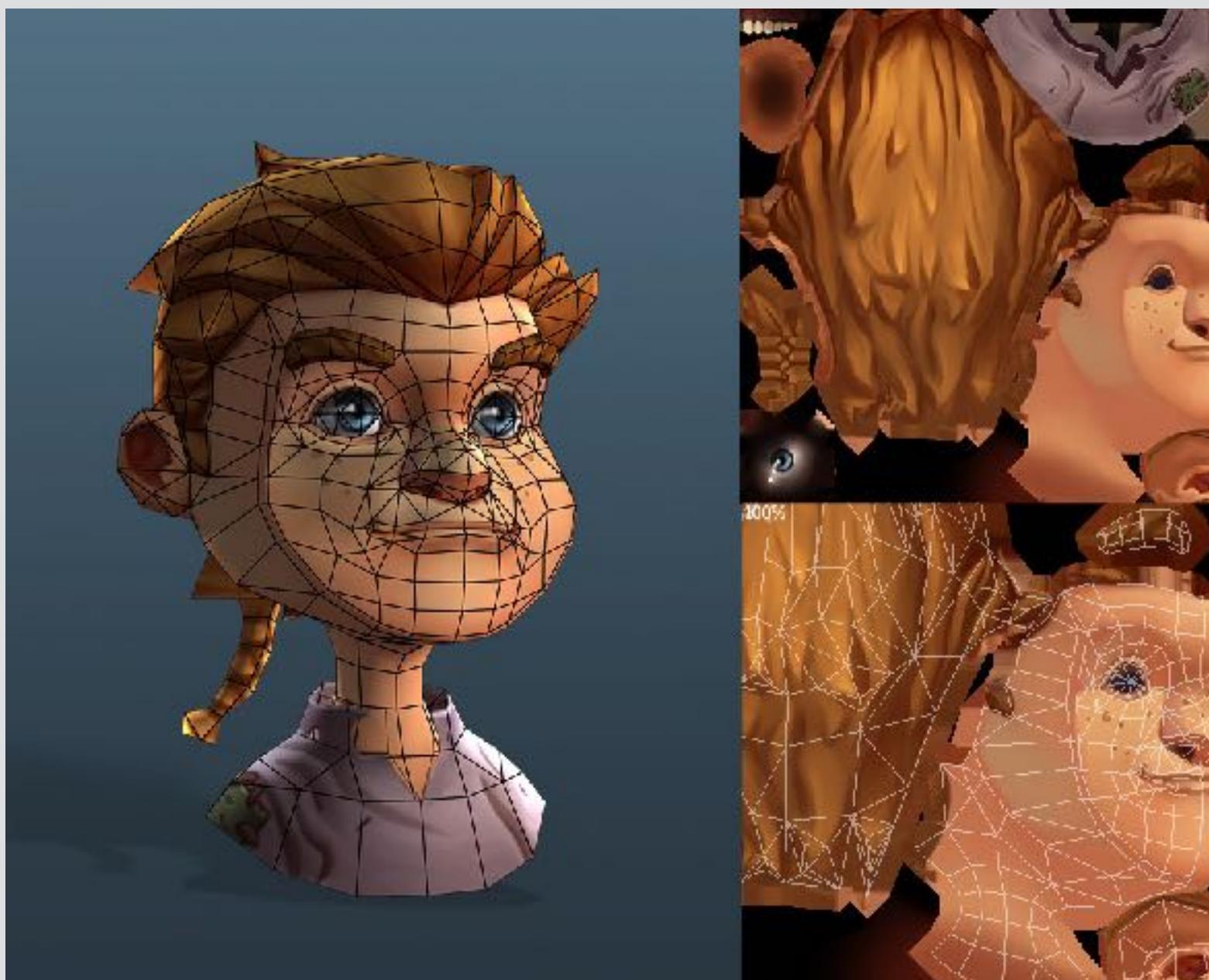
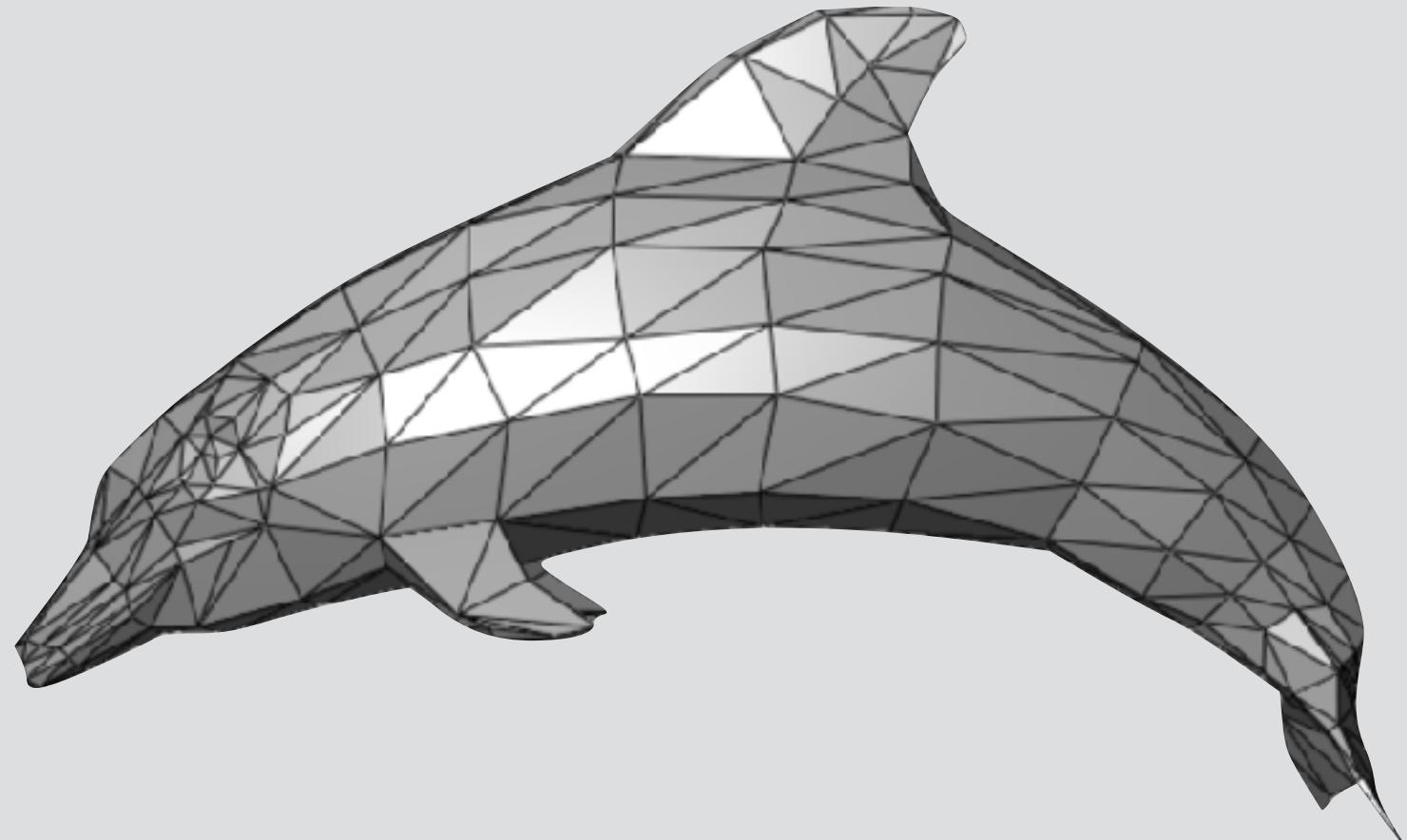
RAM

Hardware Rendering



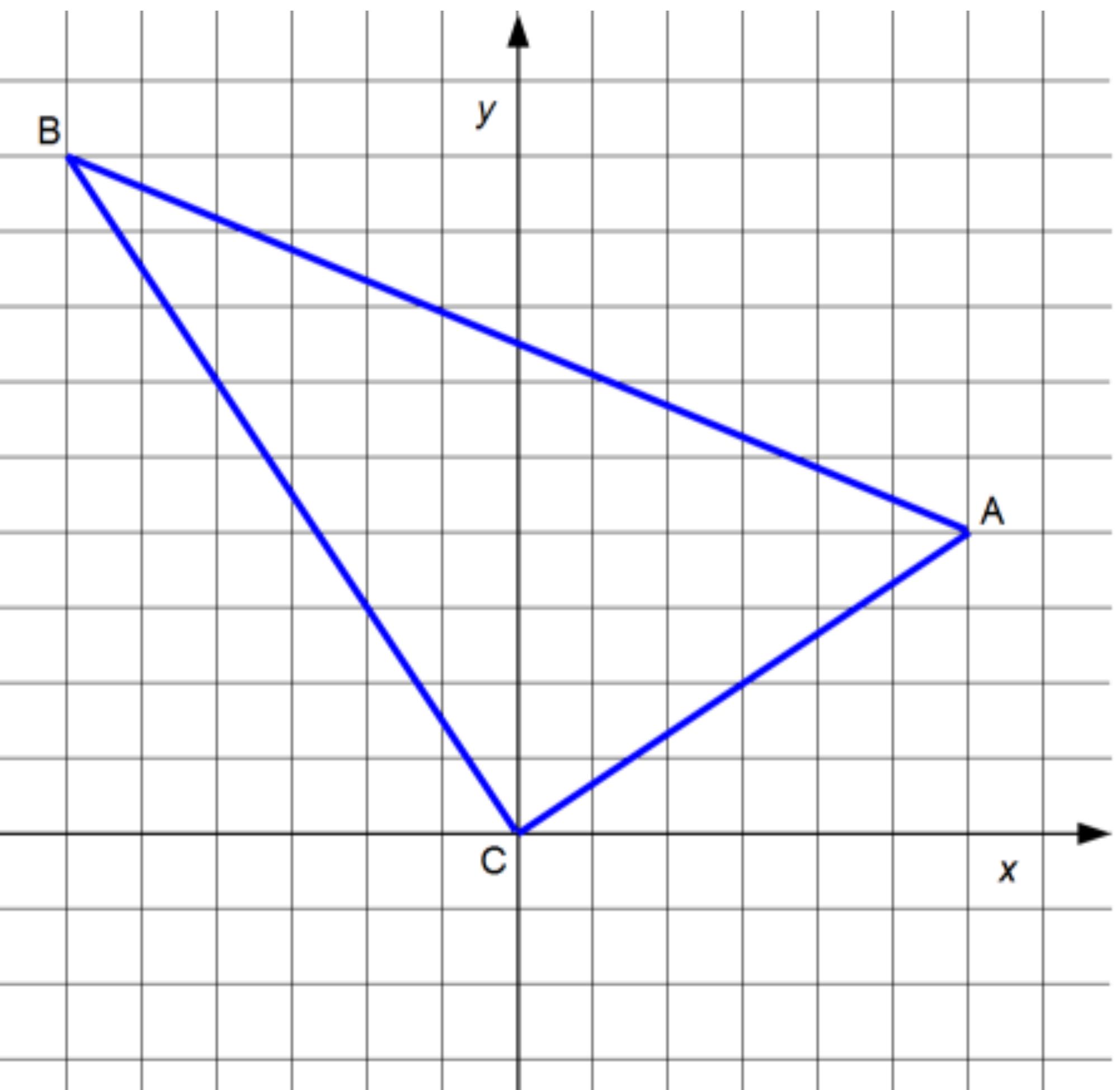
The GPU pipeline

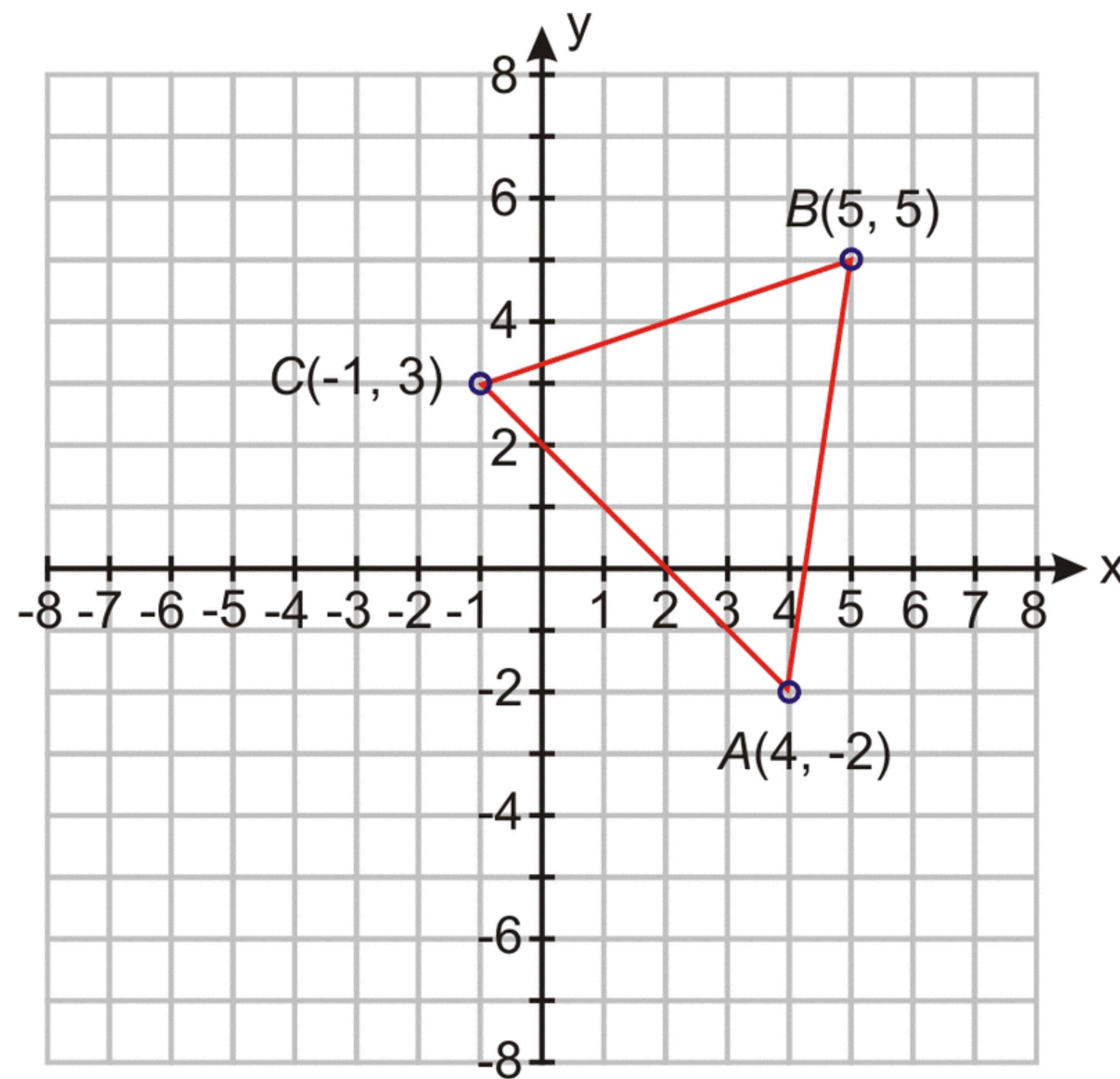
An object is a collection of polygons.

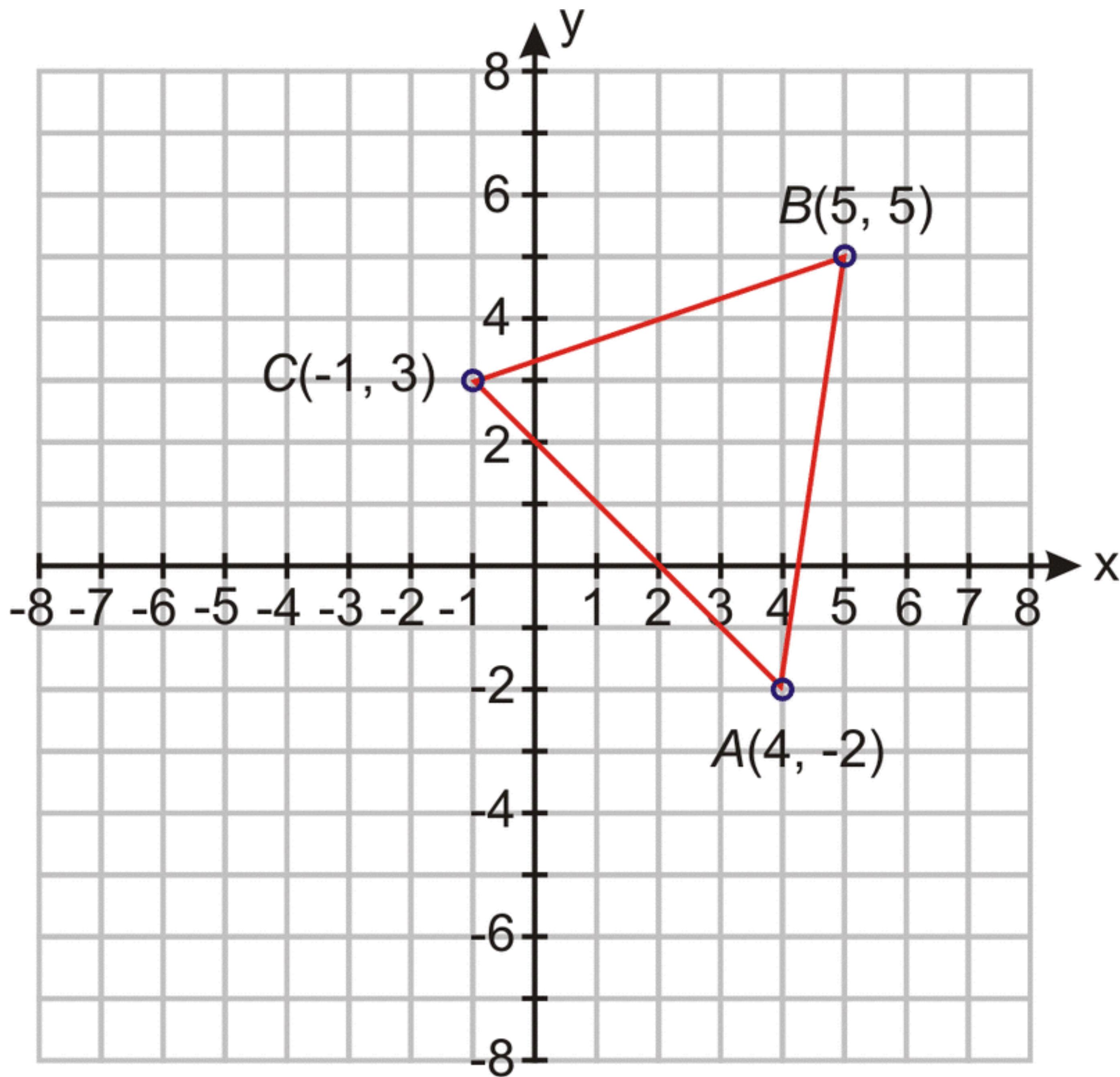


Vertex data

A polygon is defined by
points in space called vertices





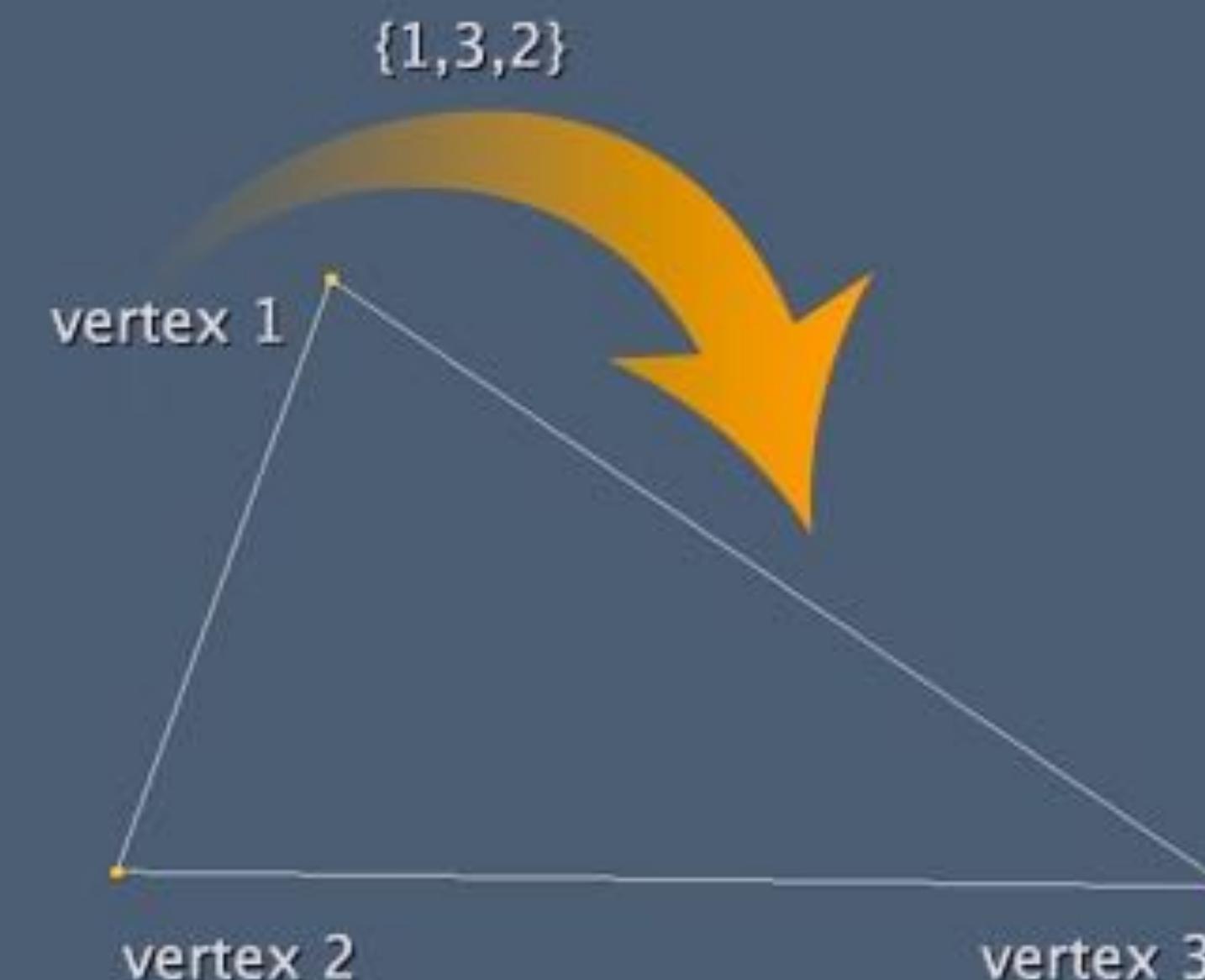


- A: (4, -2)
B: (5, 5)
C: (-1, 3)

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

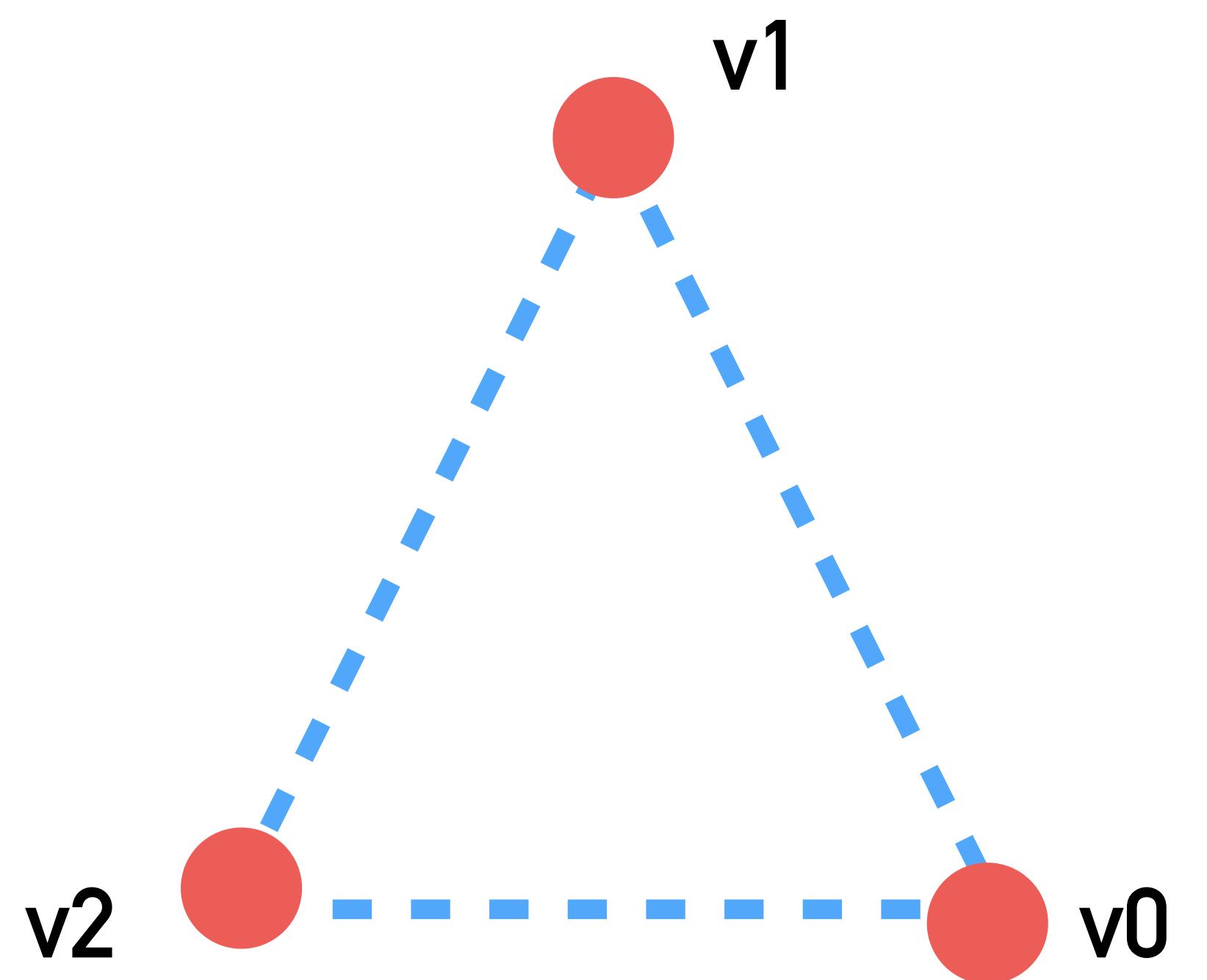


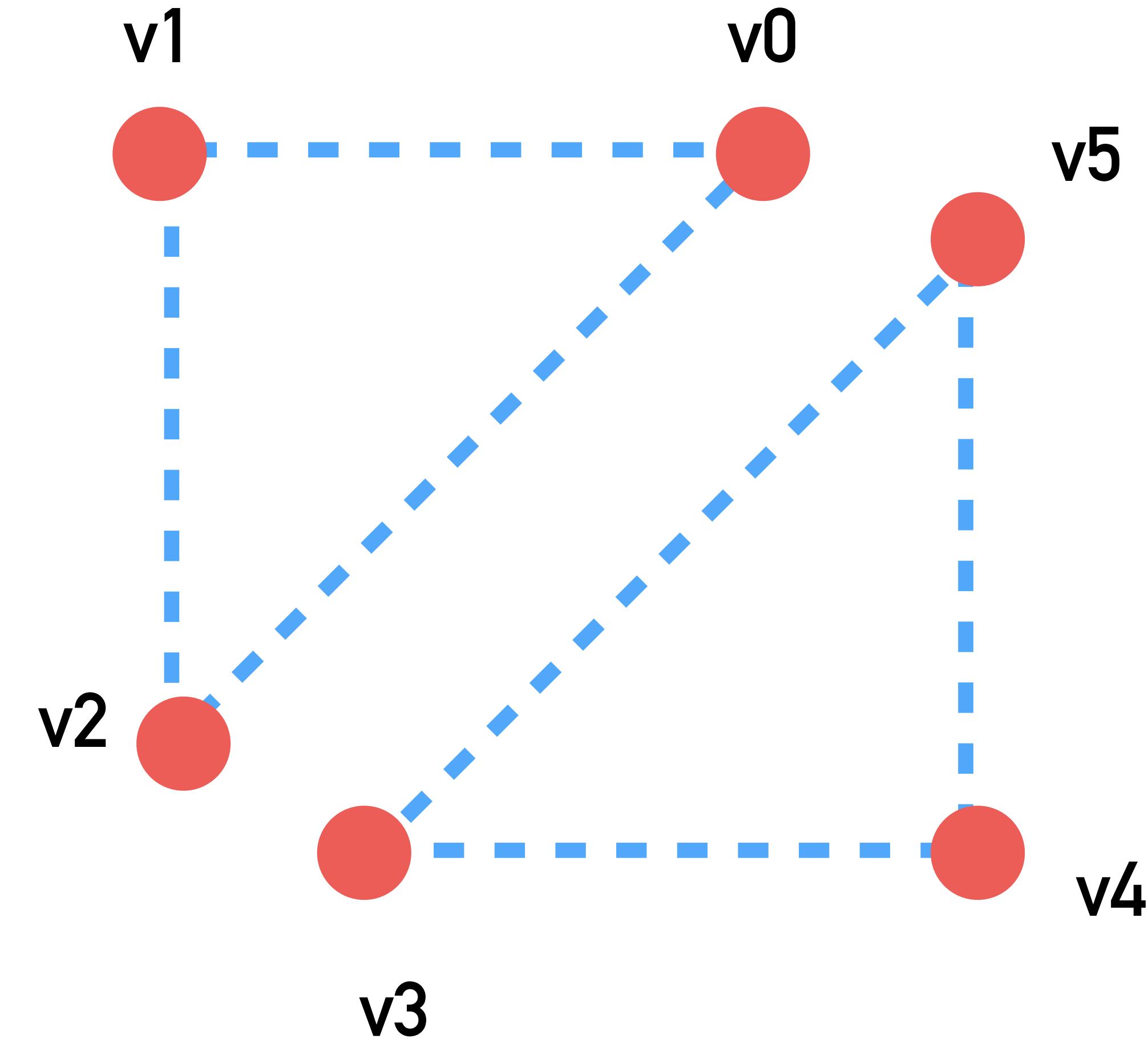
Front Facing Triangle
(CCW)



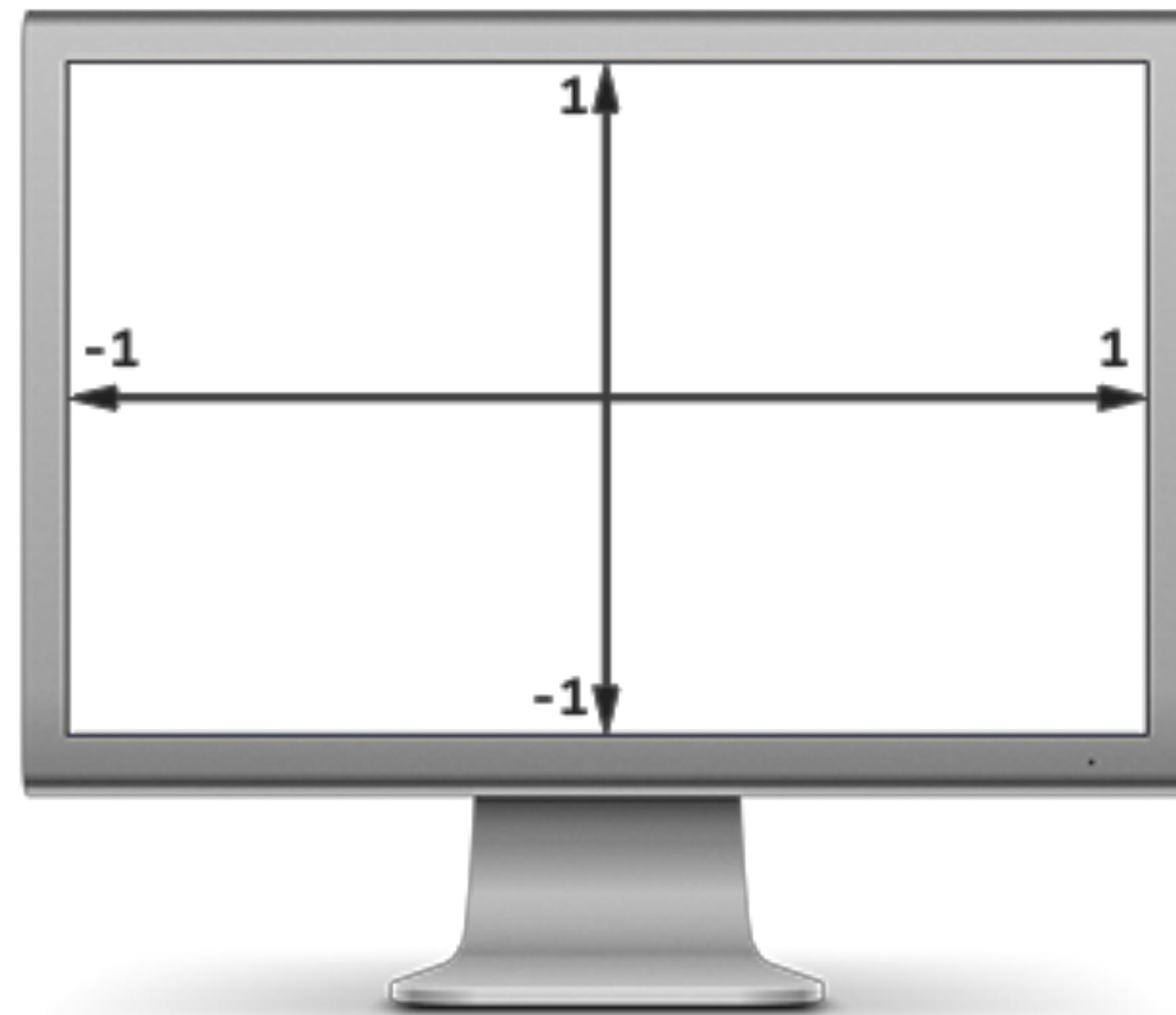
Back Facing Triangle
(CW)

Counter-clockwise order!





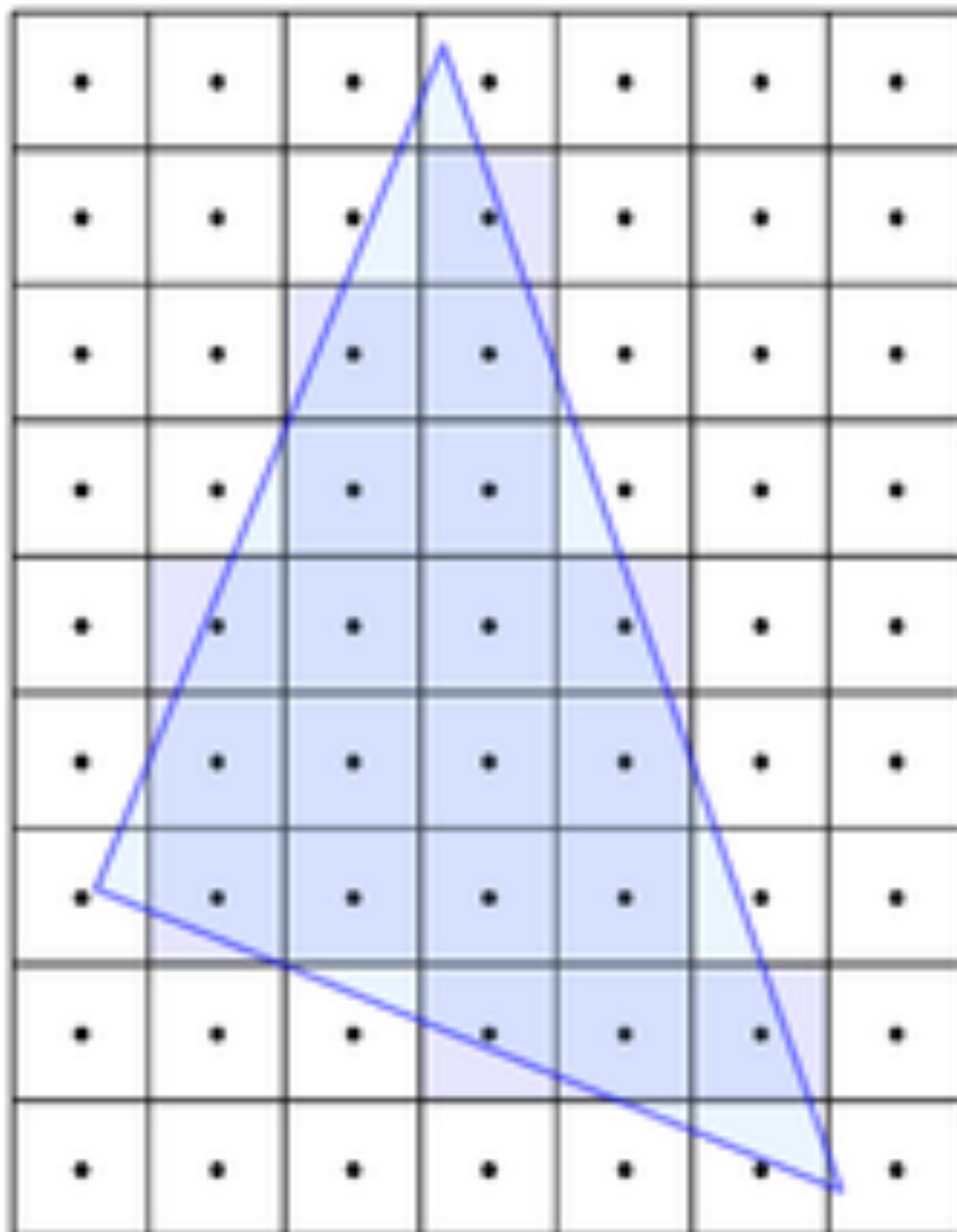
Normalized Device Coordinates



The GPU pipeline

VERTEX DATA

Rasterization

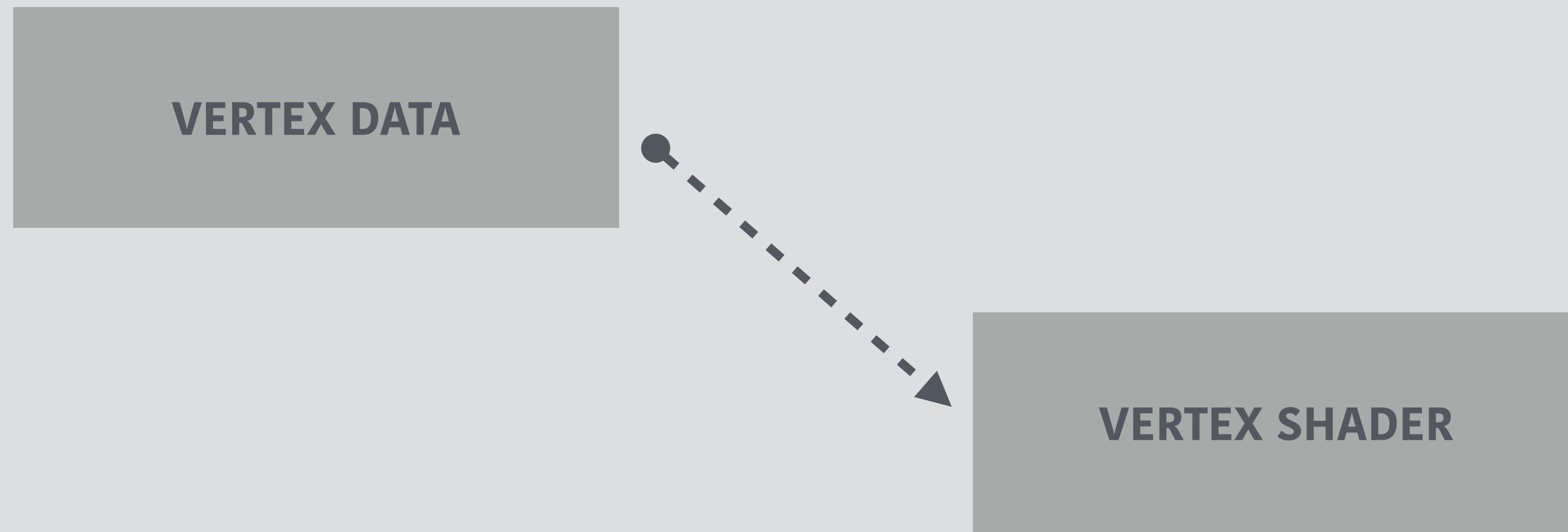


Shaders

The vertex shader

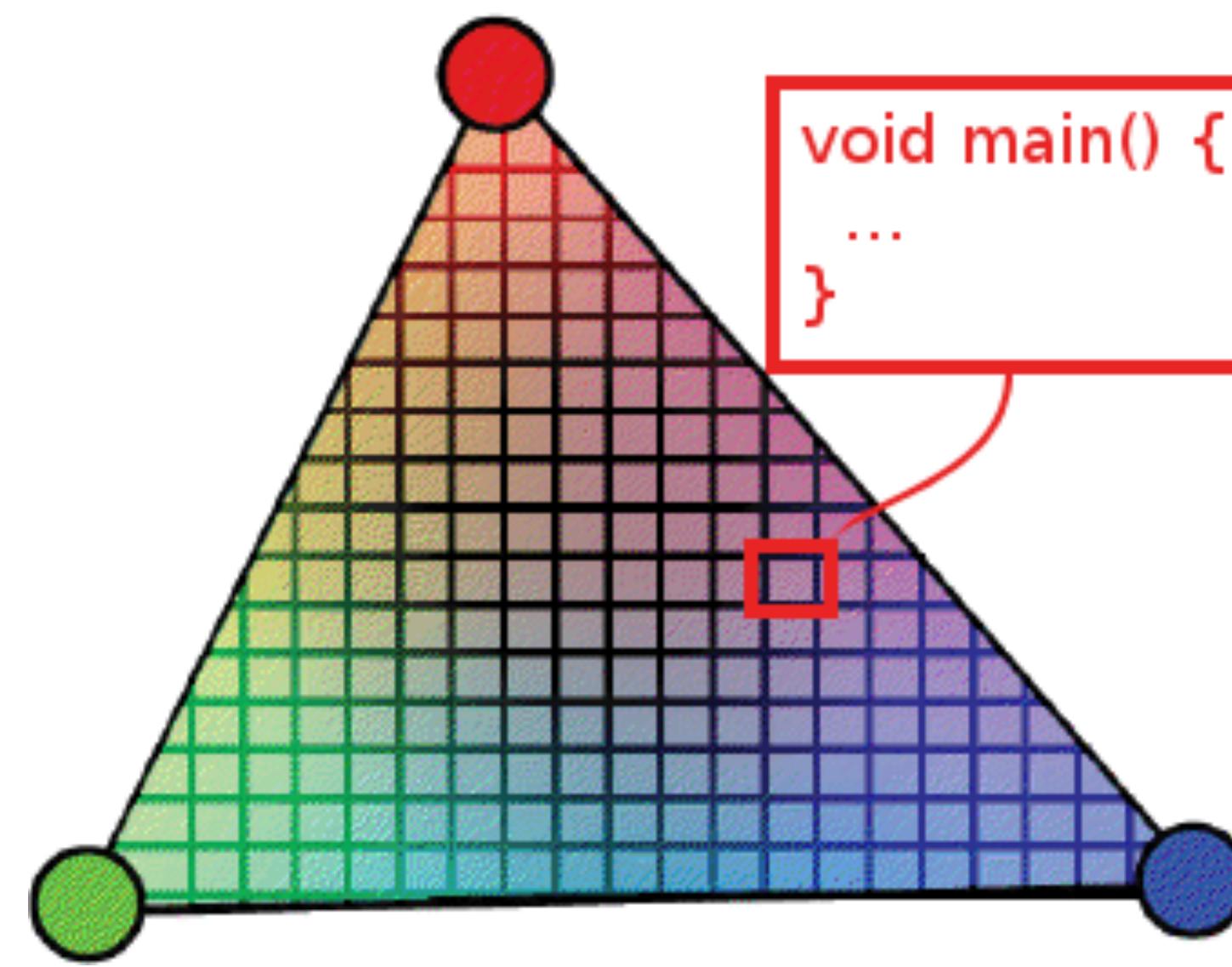
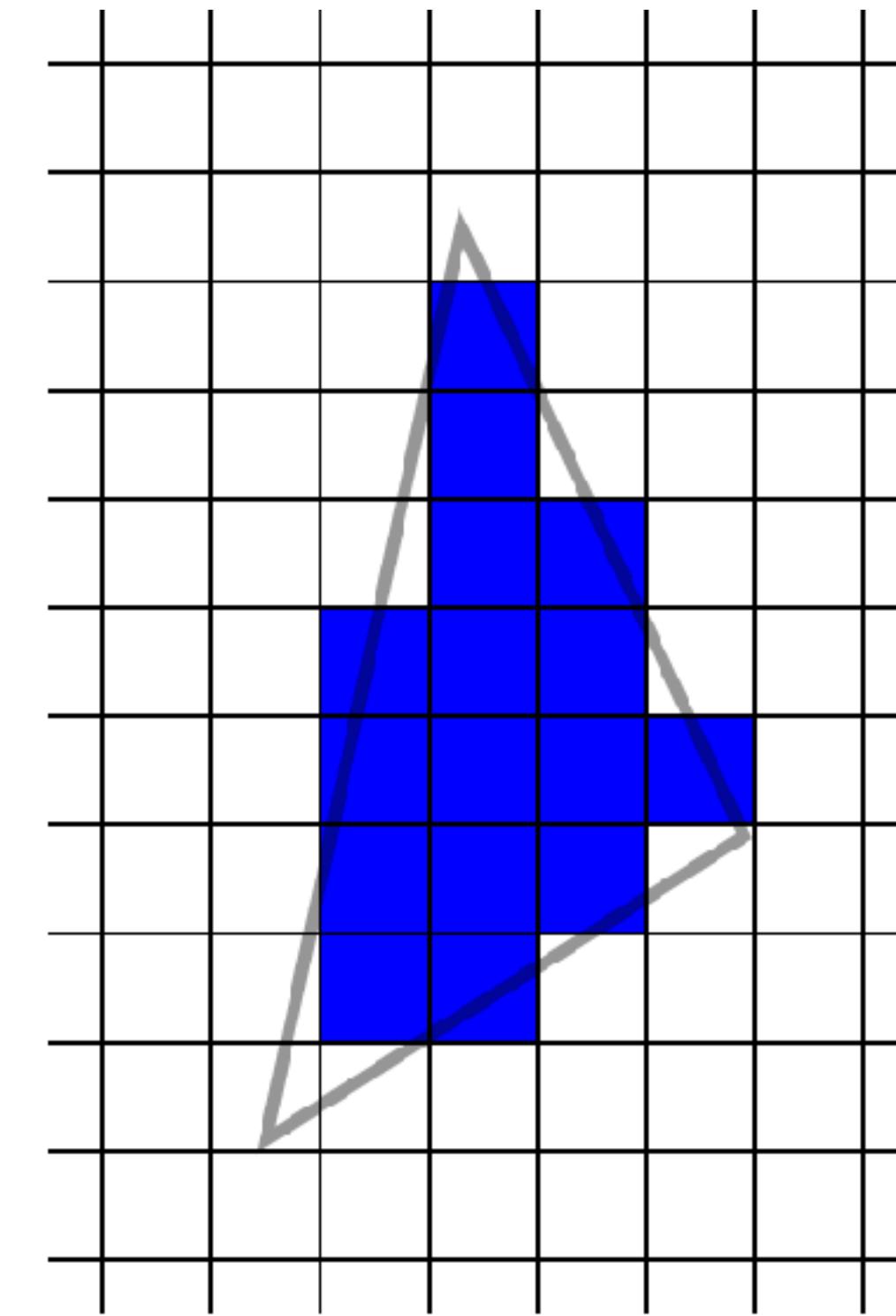
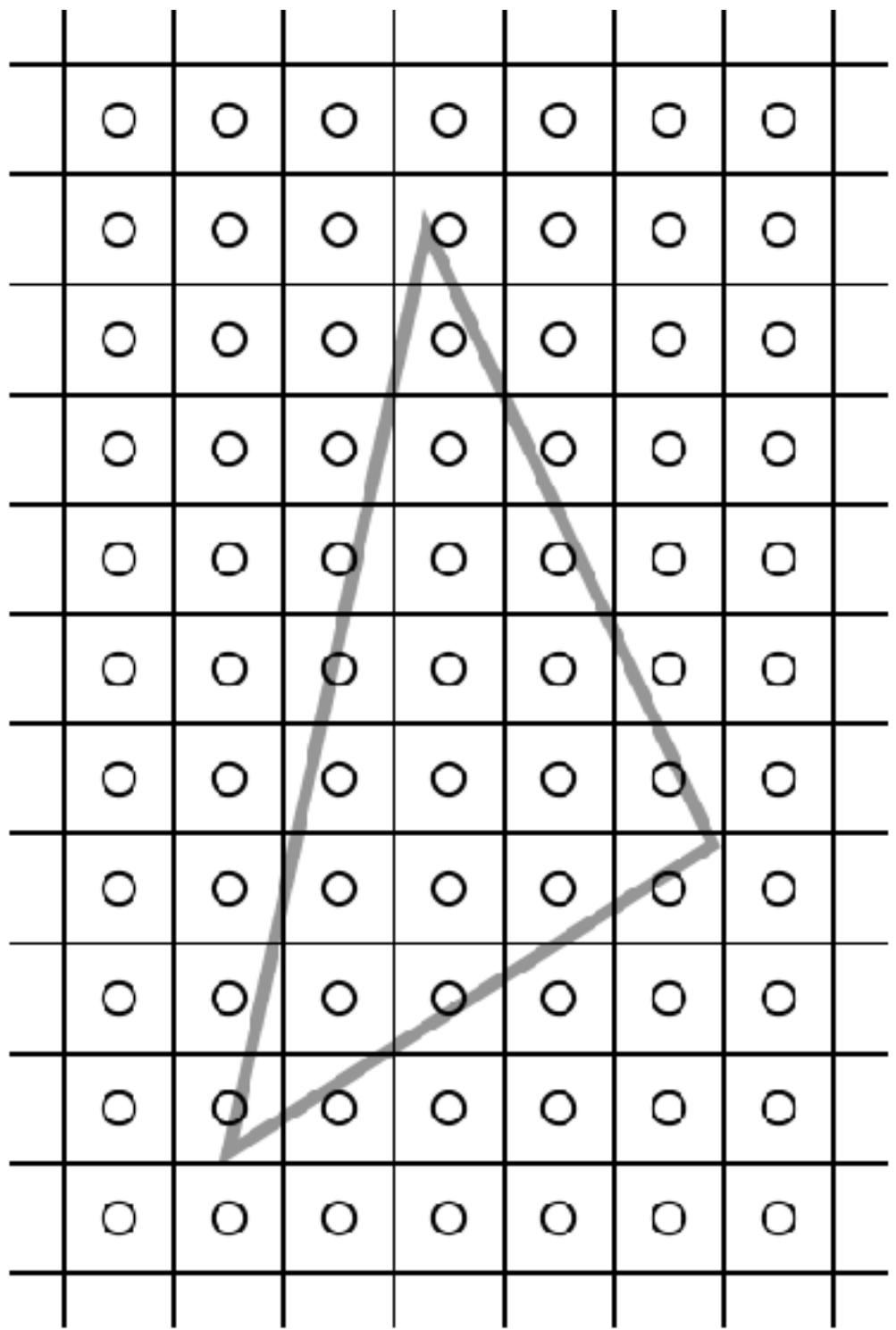
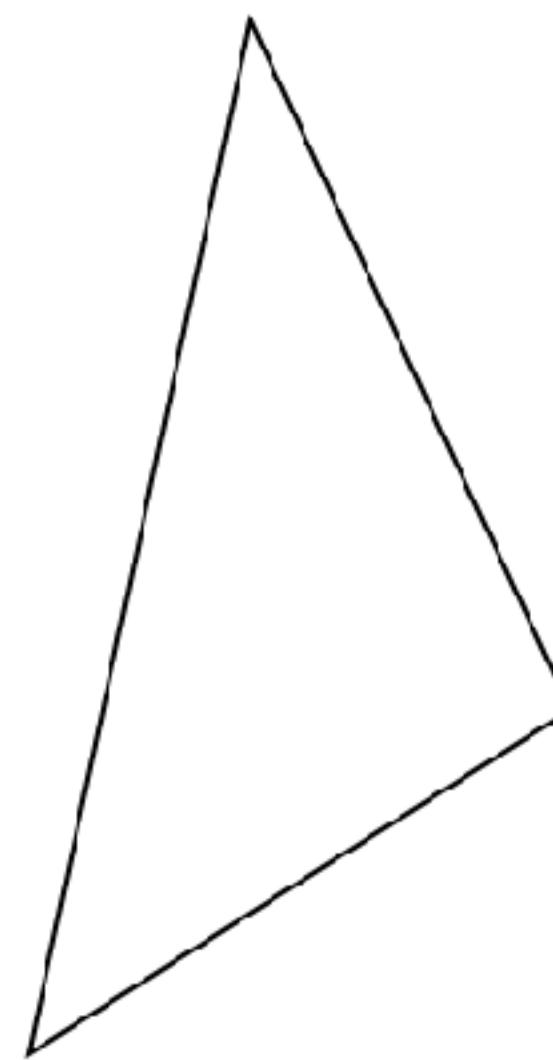
A program that runs for every vertex we render.
It transforms the attributes
(such as position, color or others)
of every vertex passed to the GPU and returns
the final position of the vertex in device
coordinates.

The GPU pipeline

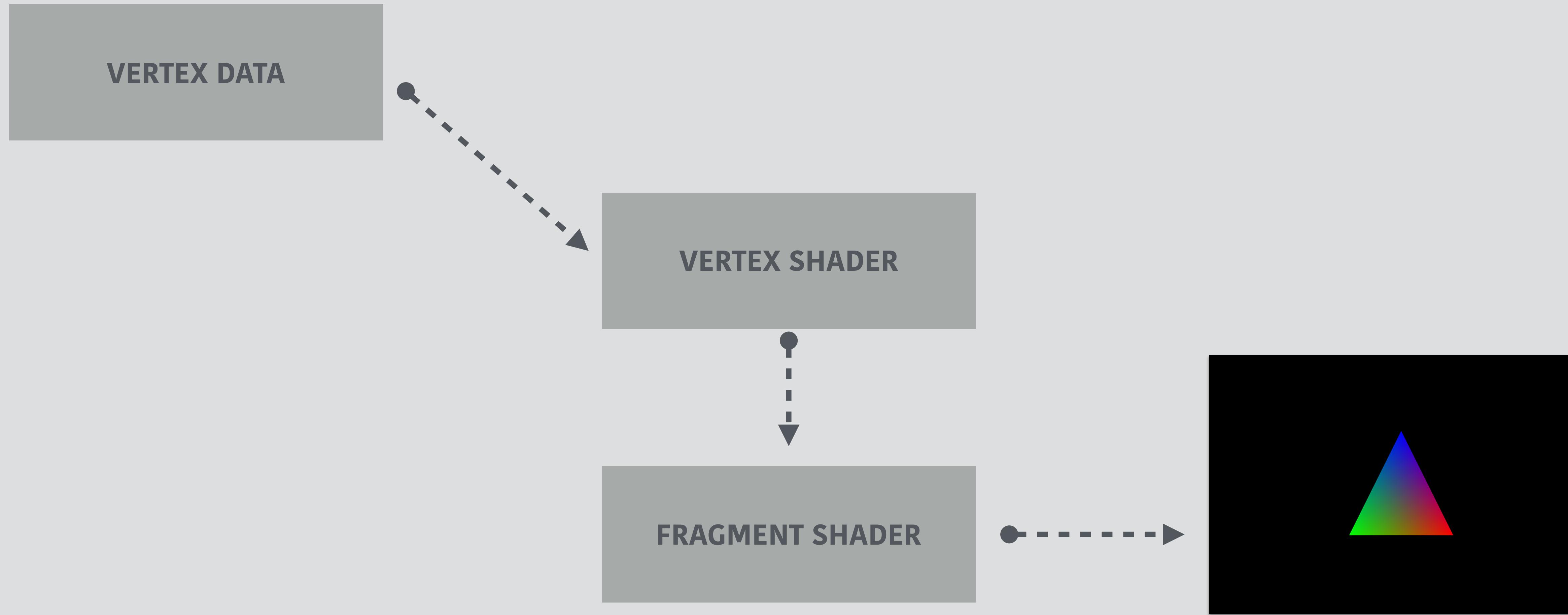


The fragment shader

A program that runs for every pixel we render. It returns **the color of each pixel** when geometry is rasterized on the **GPU**.



The GPU pipeline

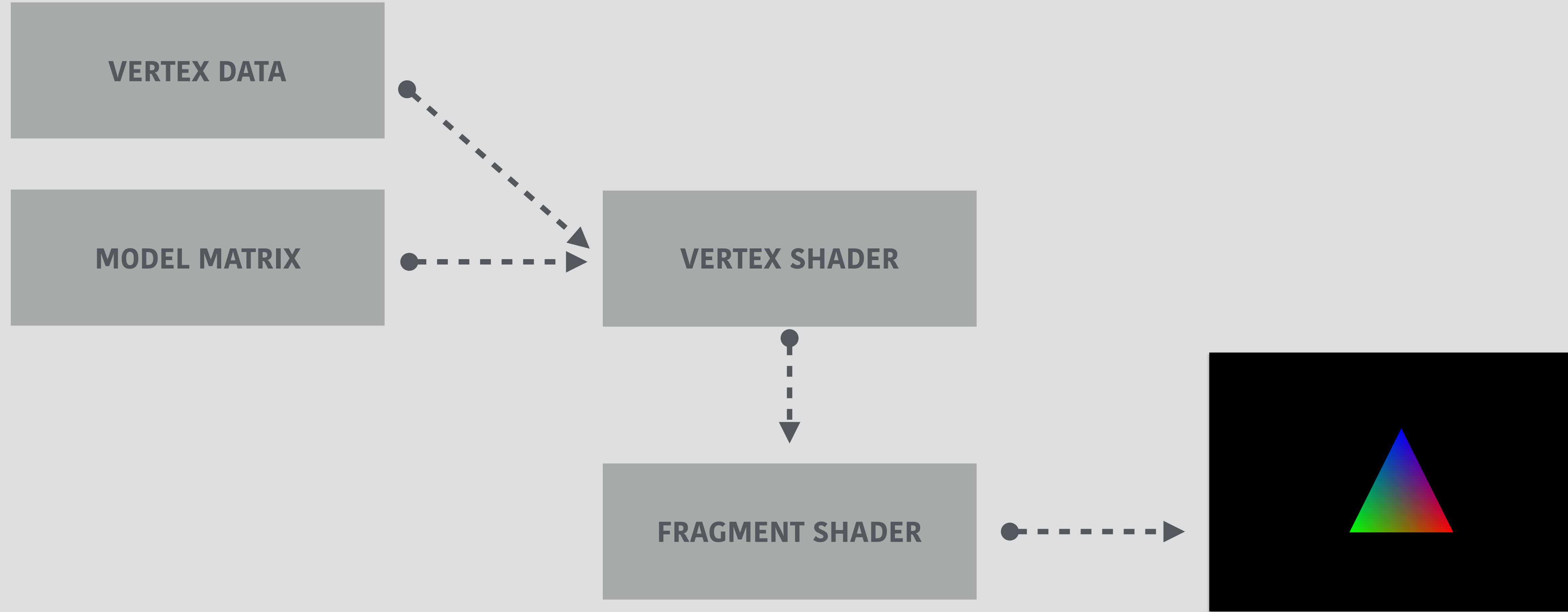


The model matrix

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

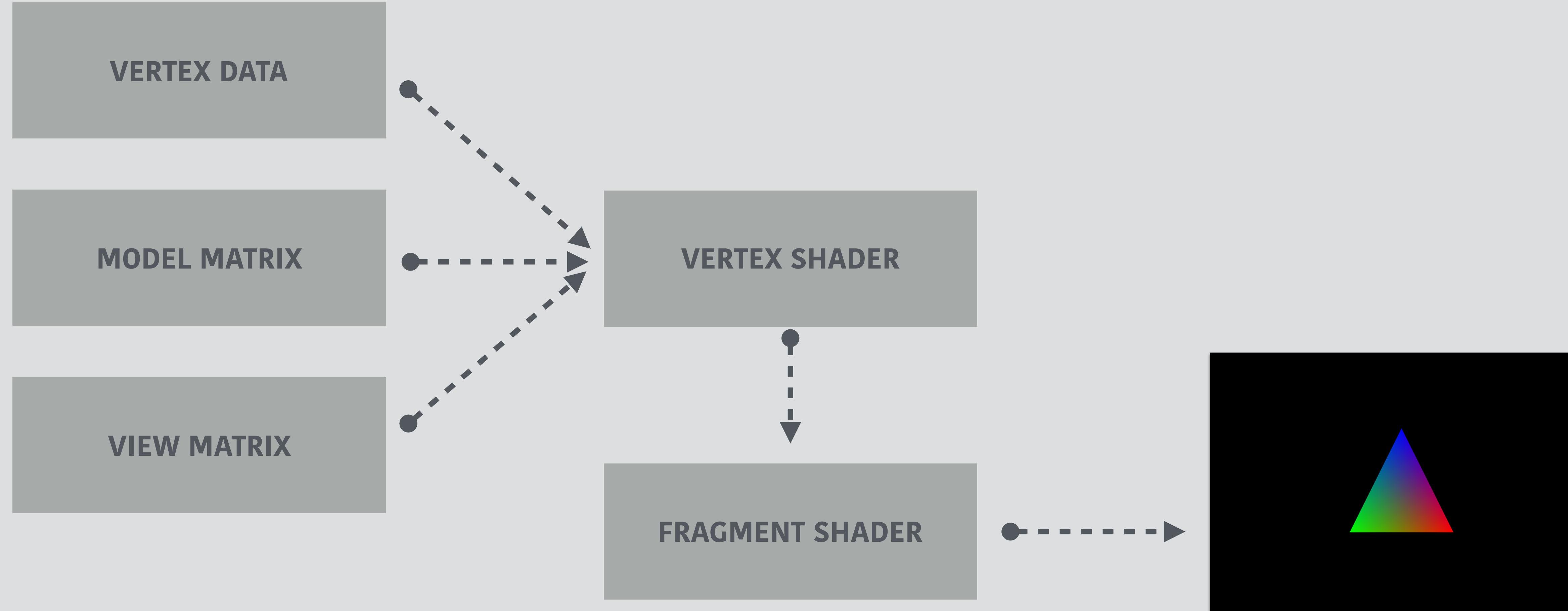


The view matrix

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

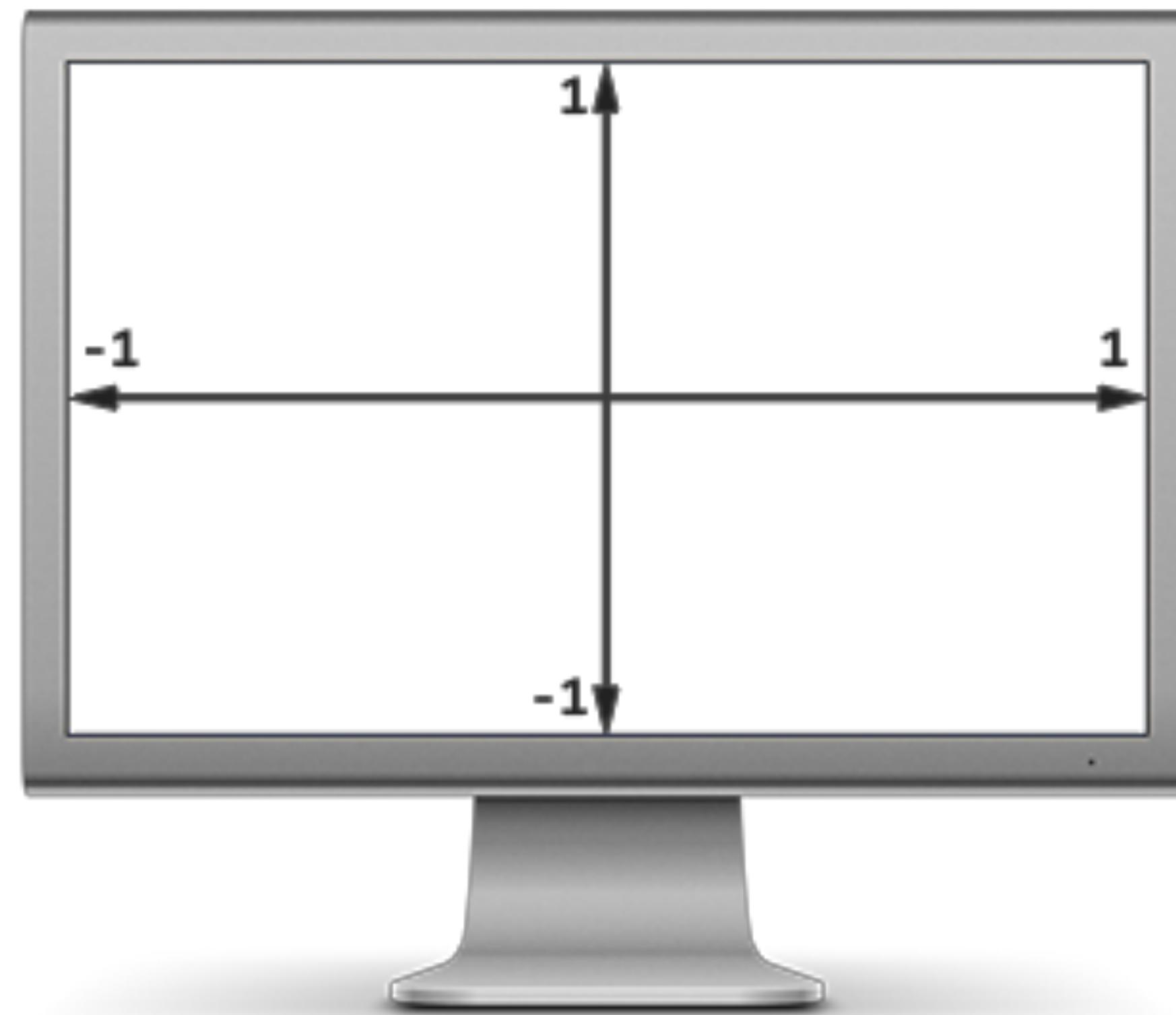
(for now we will keep it identity)

The GPU pipeline



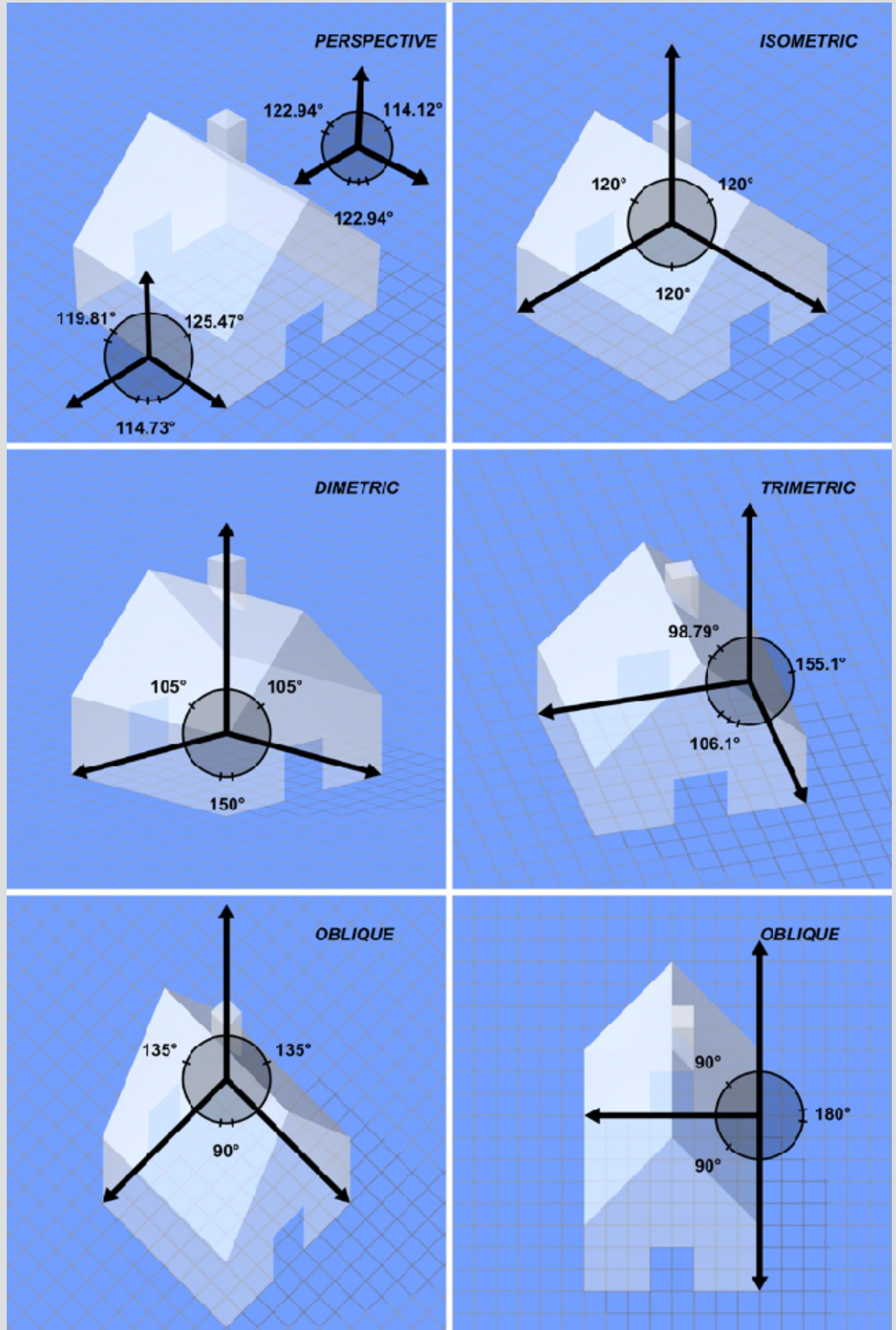
The projection matrix

Normalized Device Coordinates

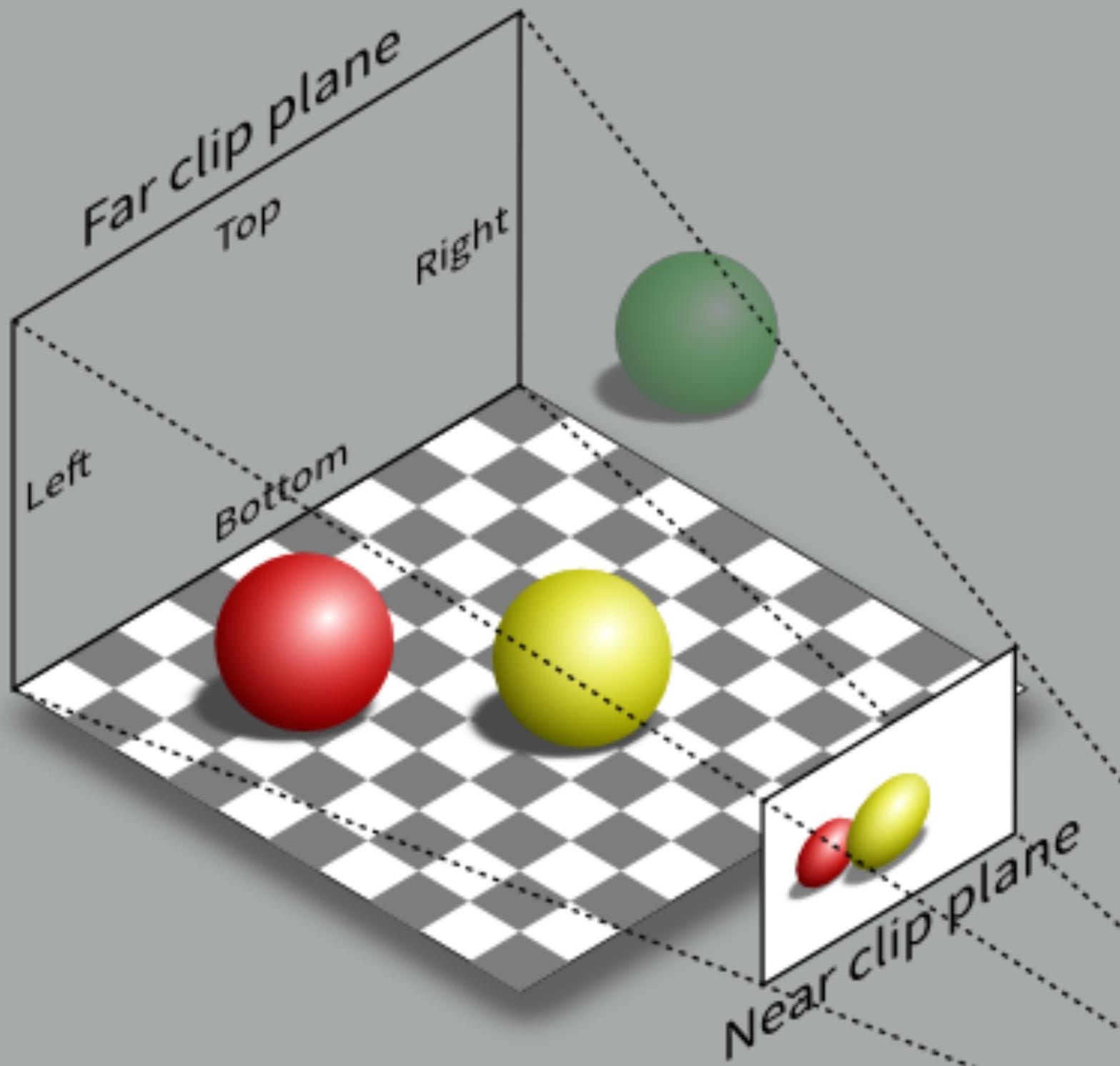


What is projection?

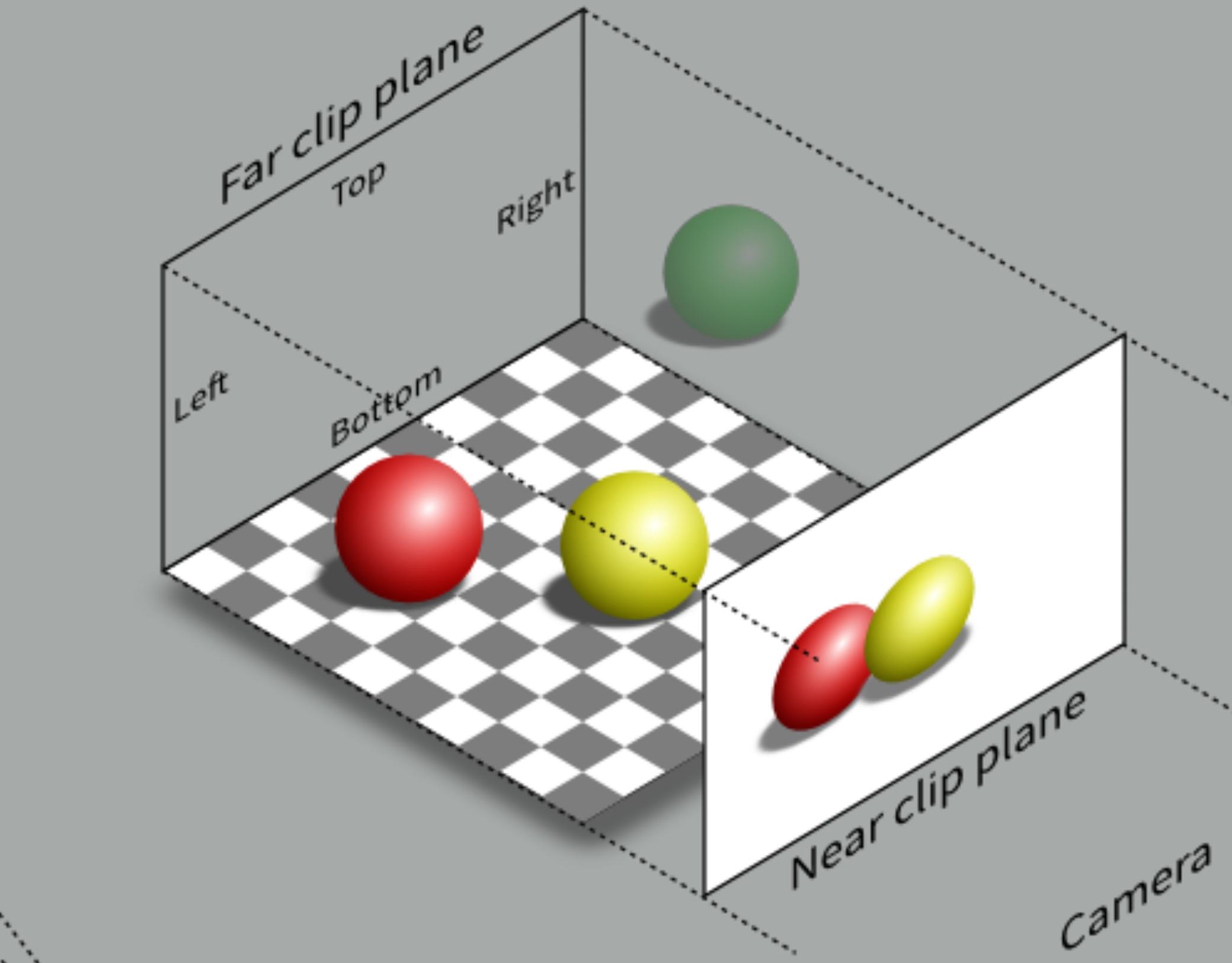
A means of representing a
three-dimensional
object in two-dimensions.



Perspective vs. Orthographic projection

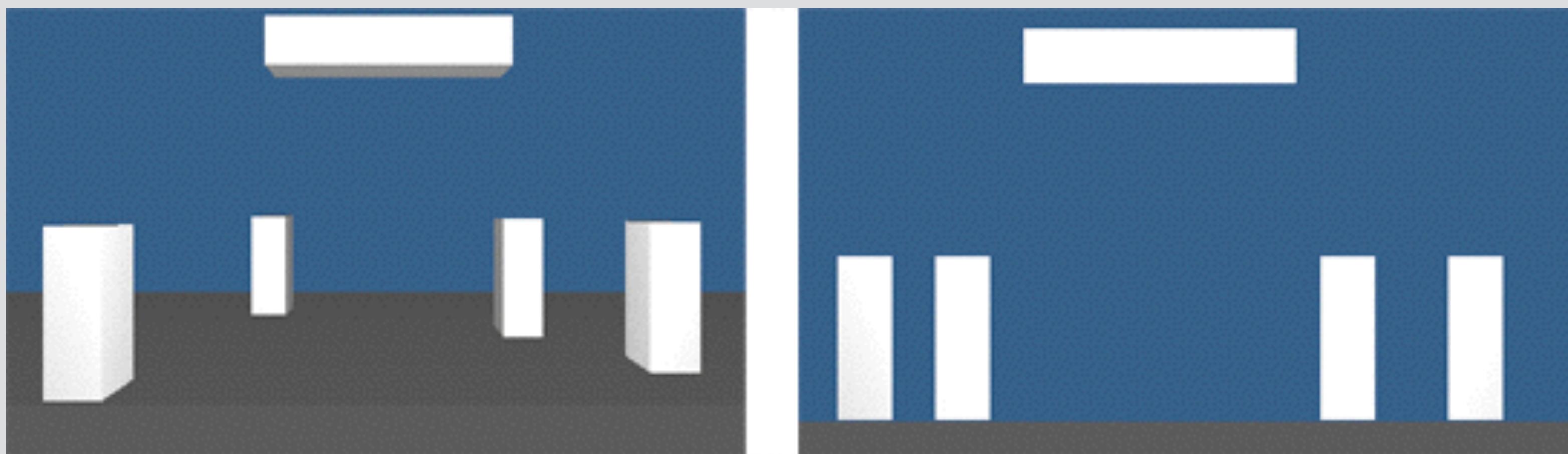
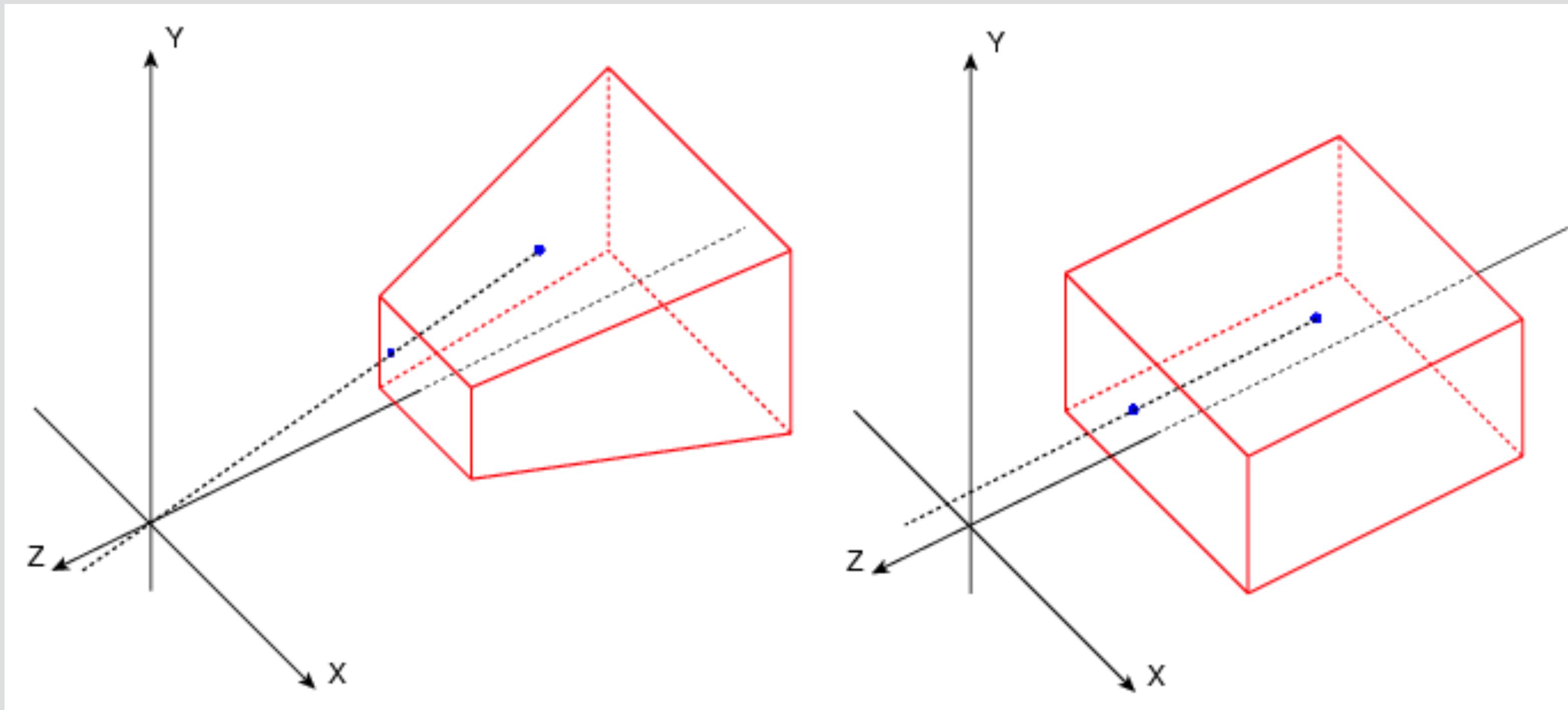


Perspective projection (P)

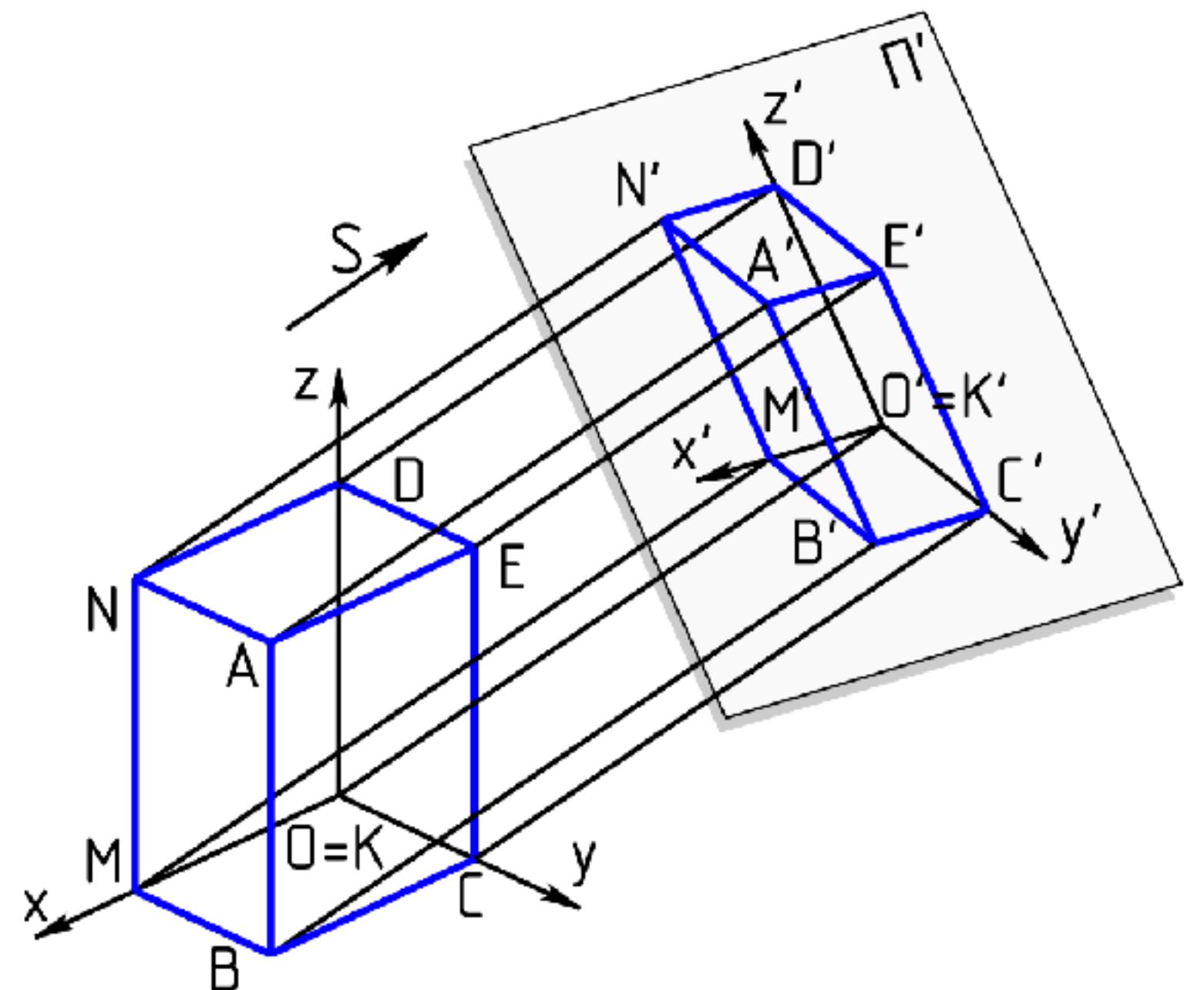


Orthographic projection (O)

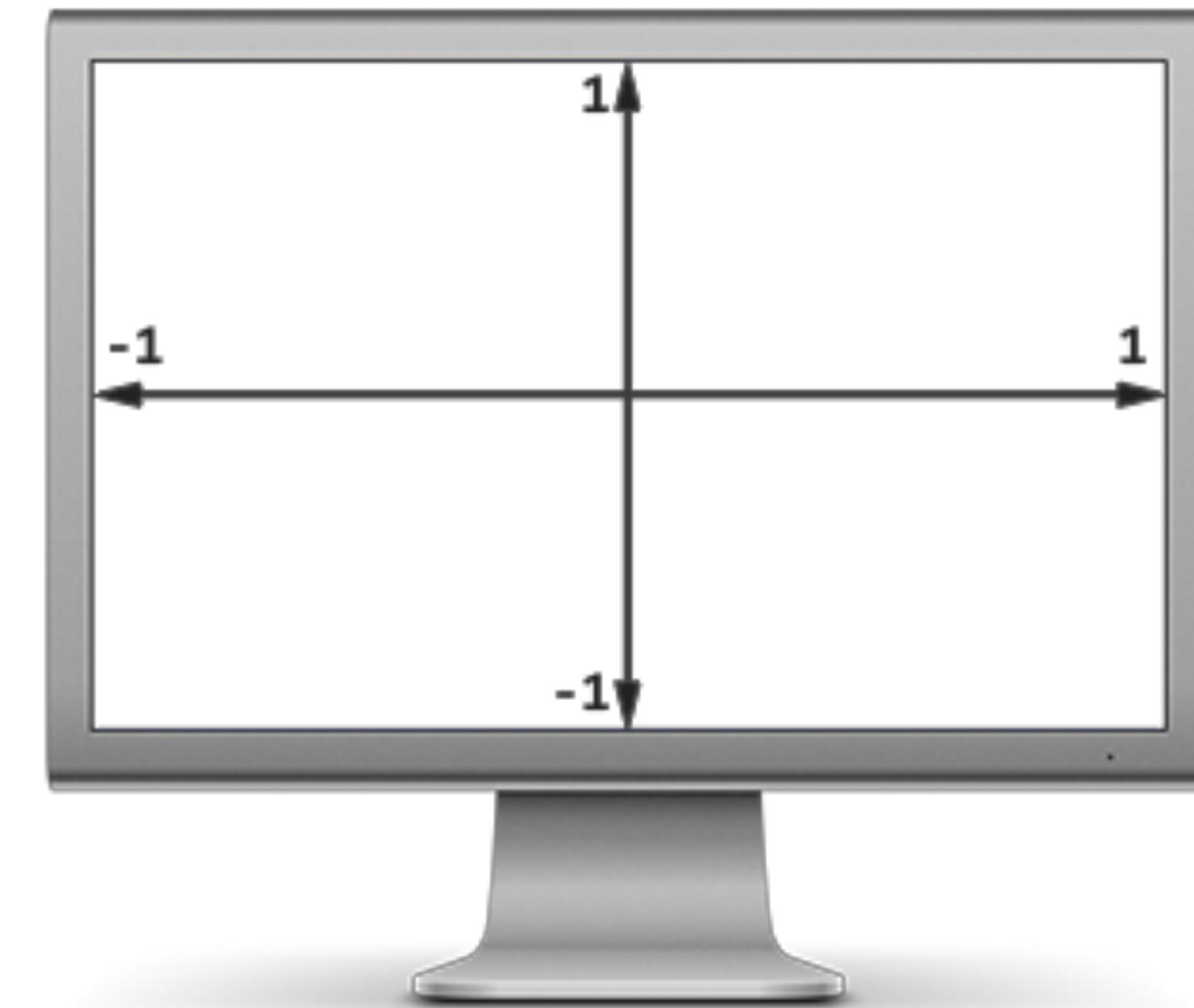
Perspective vs. Orthographic projection



Projection matrix



NORMALIZED DEVICE COORDINATES

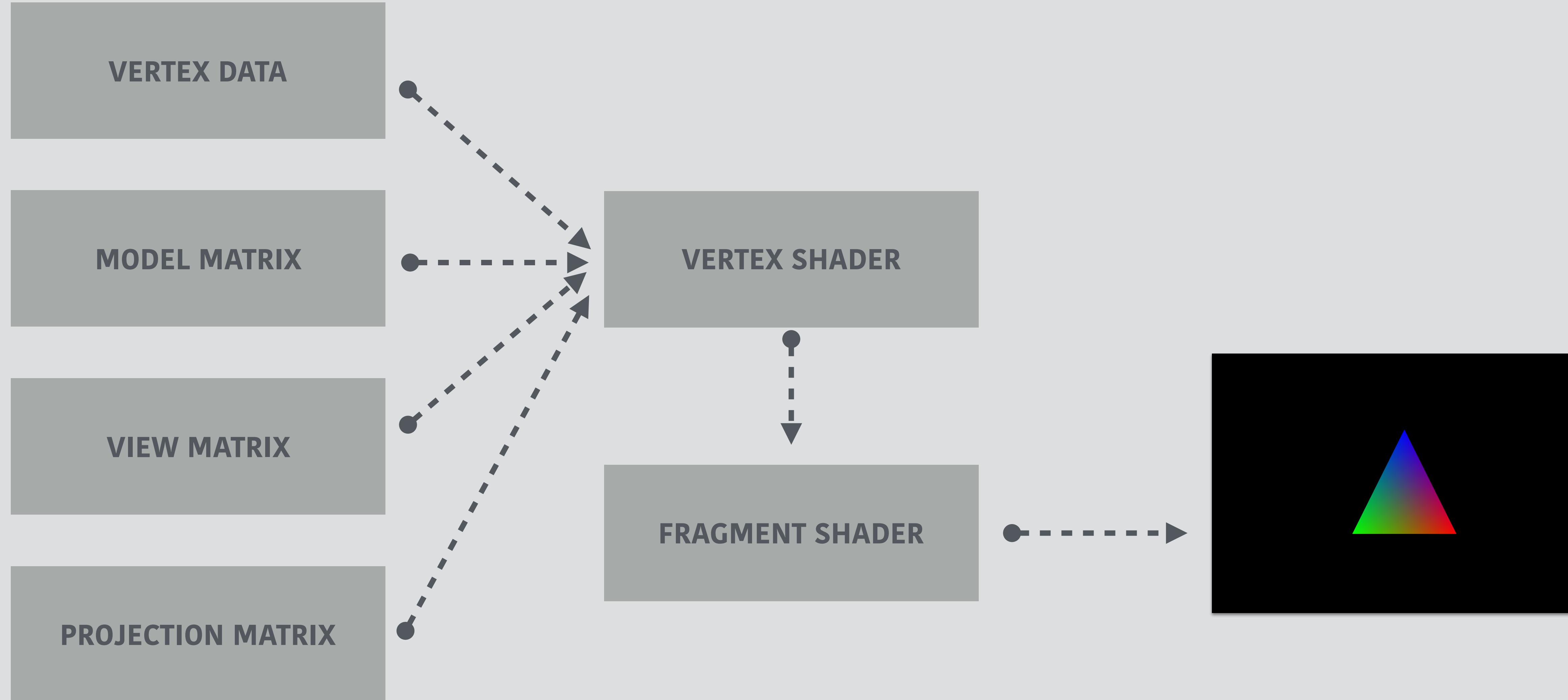


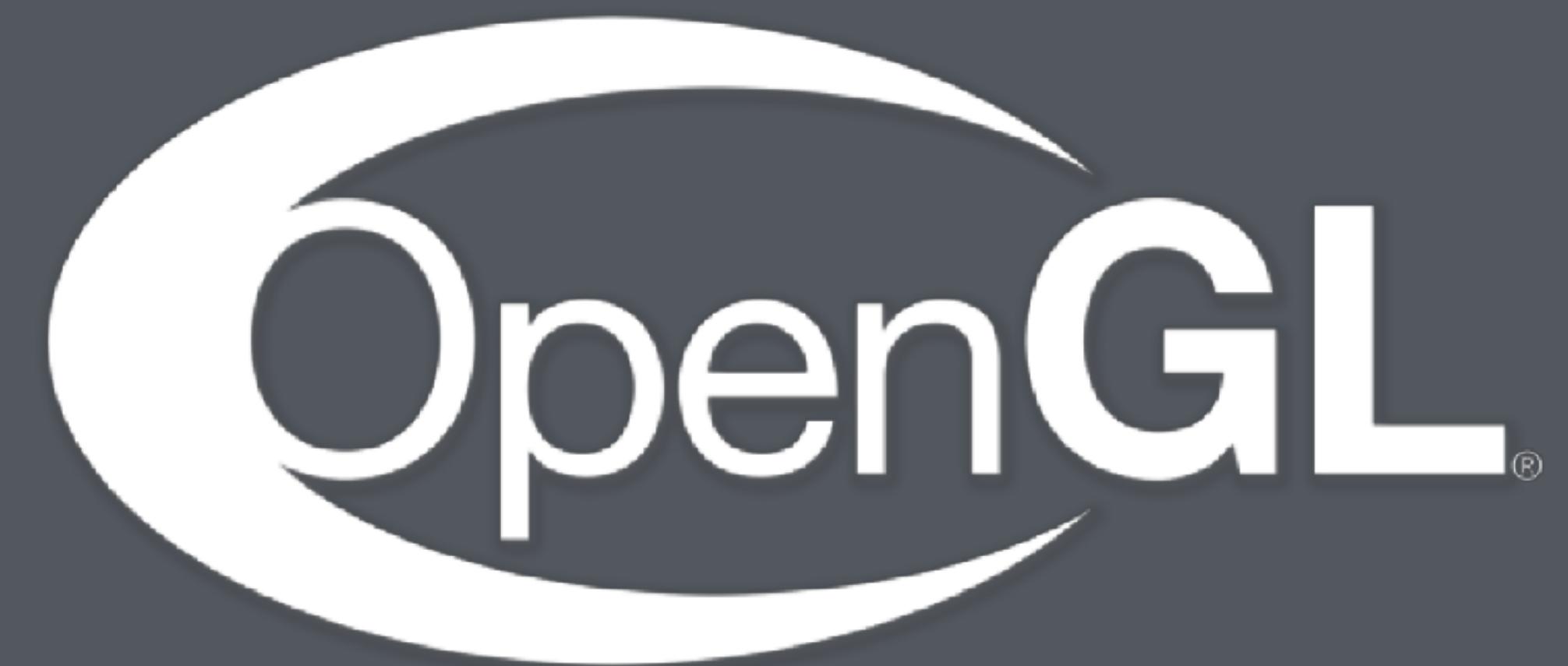
Vertex position * projection matrix = screen vertex position
in normalized device coordinates

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.

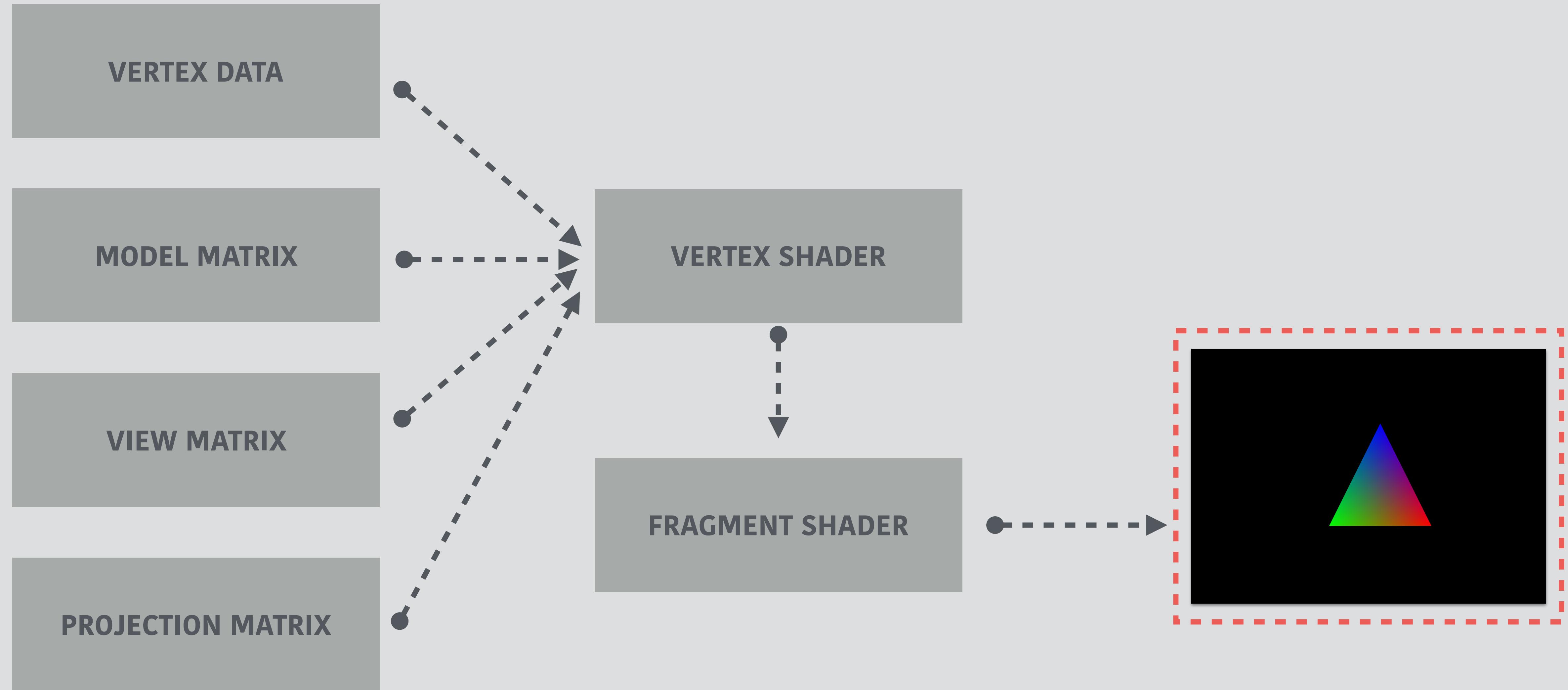
The GPU pipeline





The setup
(happens only once!)

The GPU pipeline

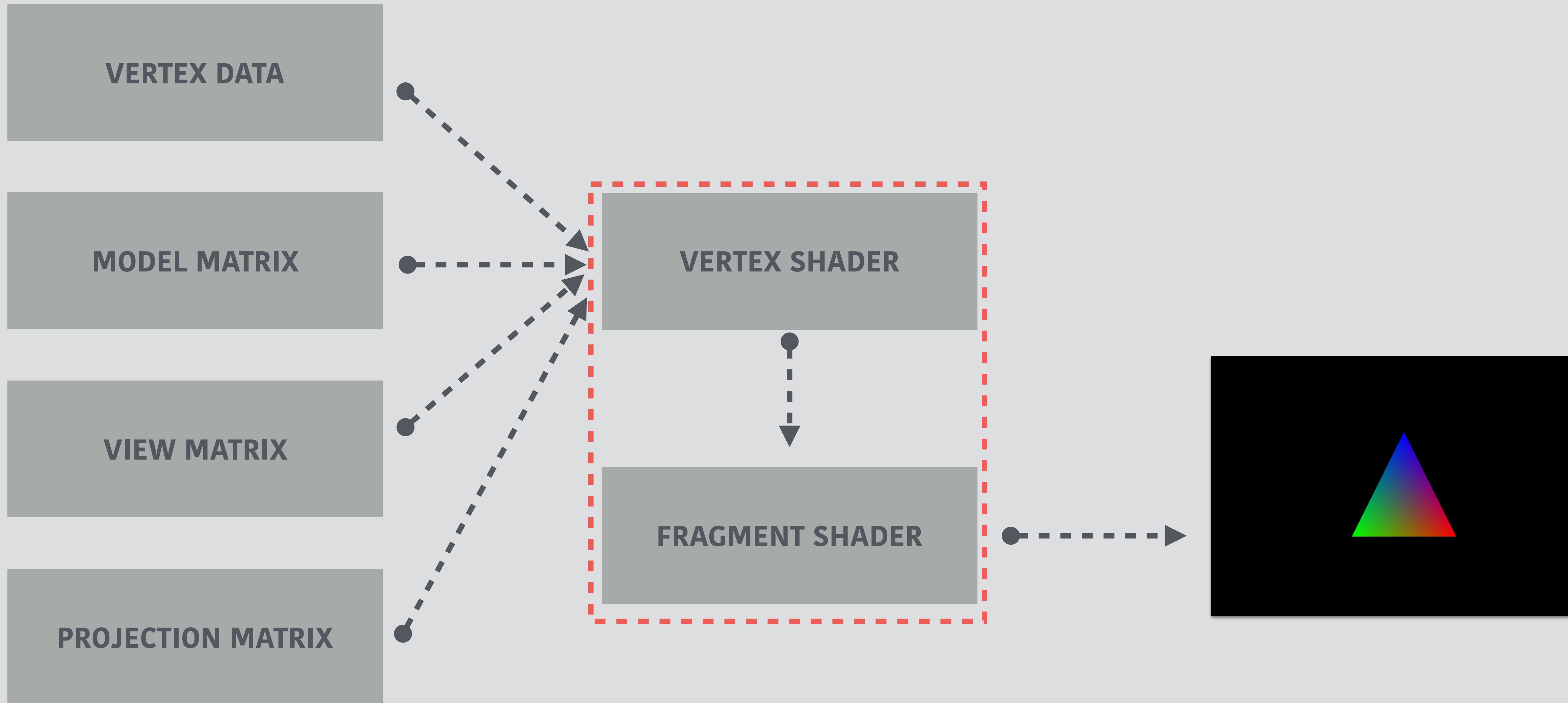


```
void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);
```

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

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

The GPU pipeline



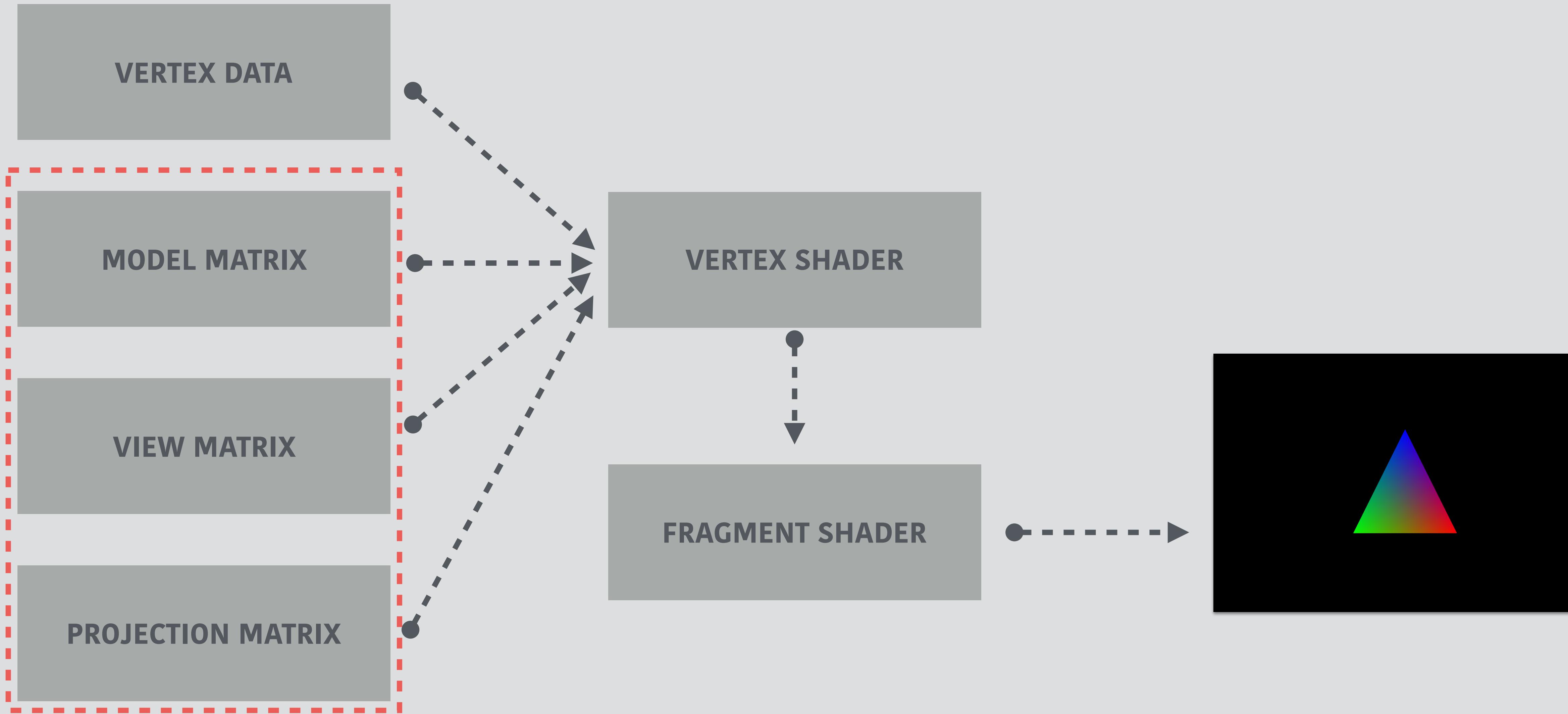
The **ShaderProgram** class.

```
#include "ShaderProgram.h"

// ...

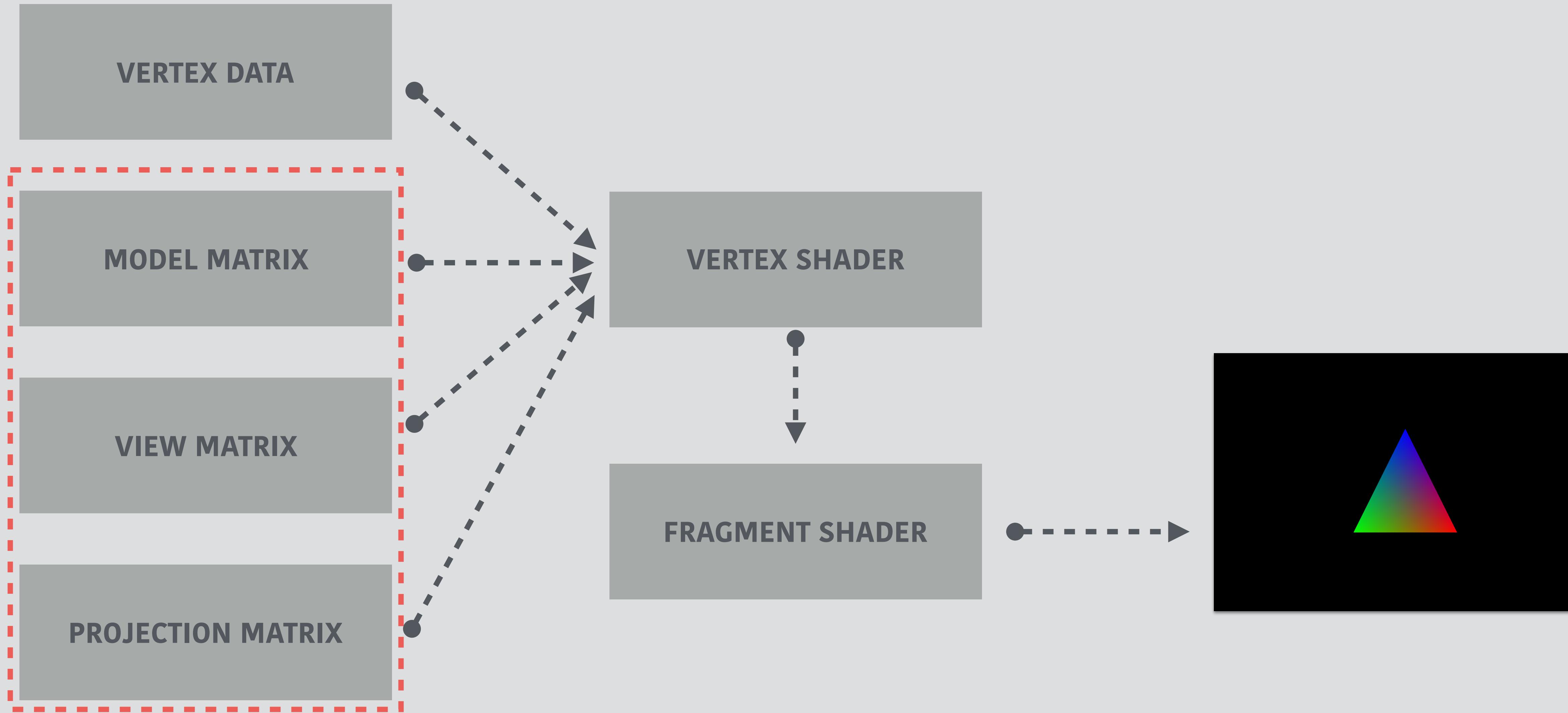
ShaderProgram program;
program.Load(RESOURCE_FOLDER"vertex.glsL", RESOURCE_FOLDER"fragment.glsL");
```

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);
```

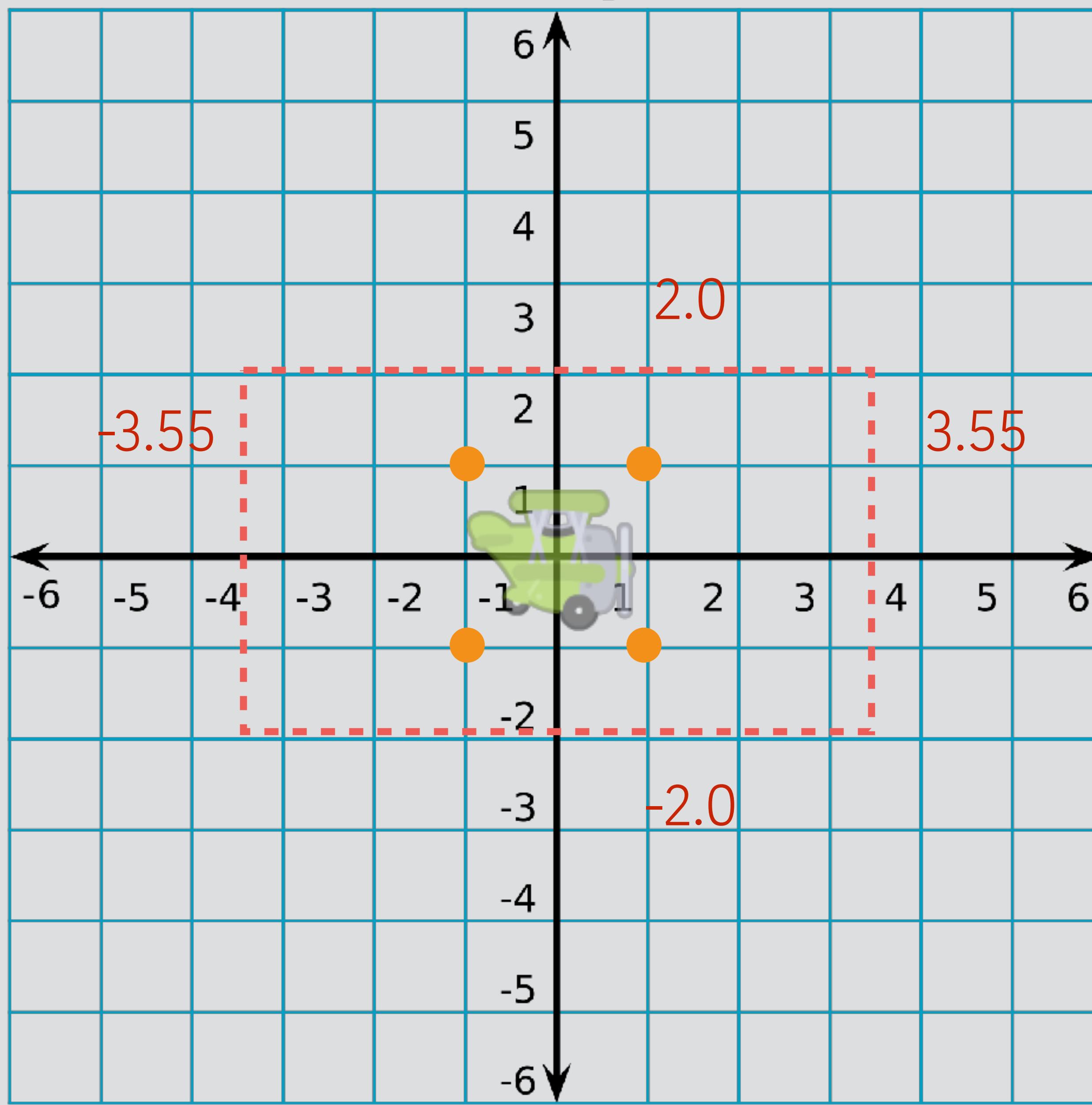
```
glm::mat4 glm::ortho(float left, float right, float bottom,  
float top, float near, float far);
```

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);
```

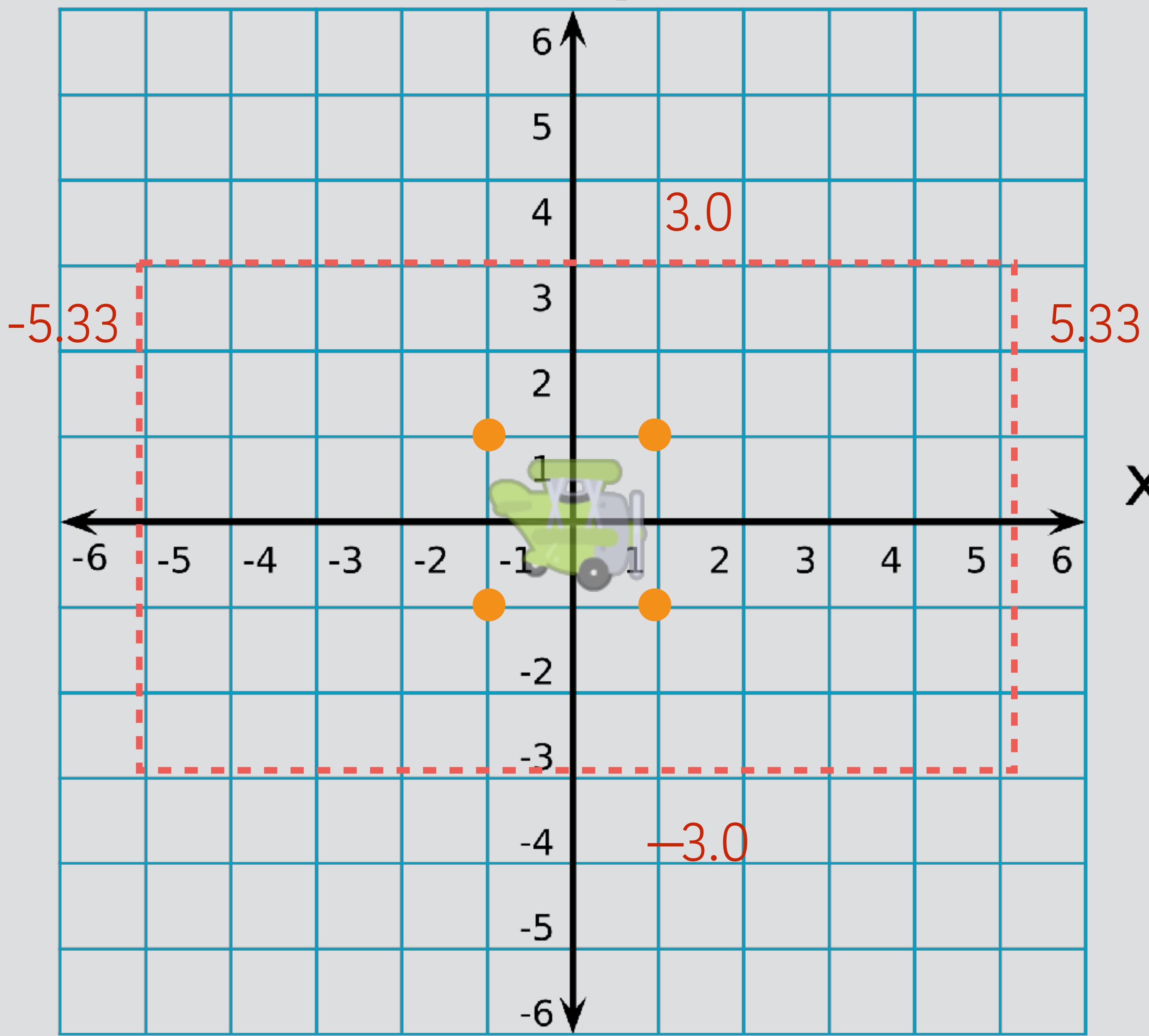
y-axis

x-axis



y-axis

x-axis

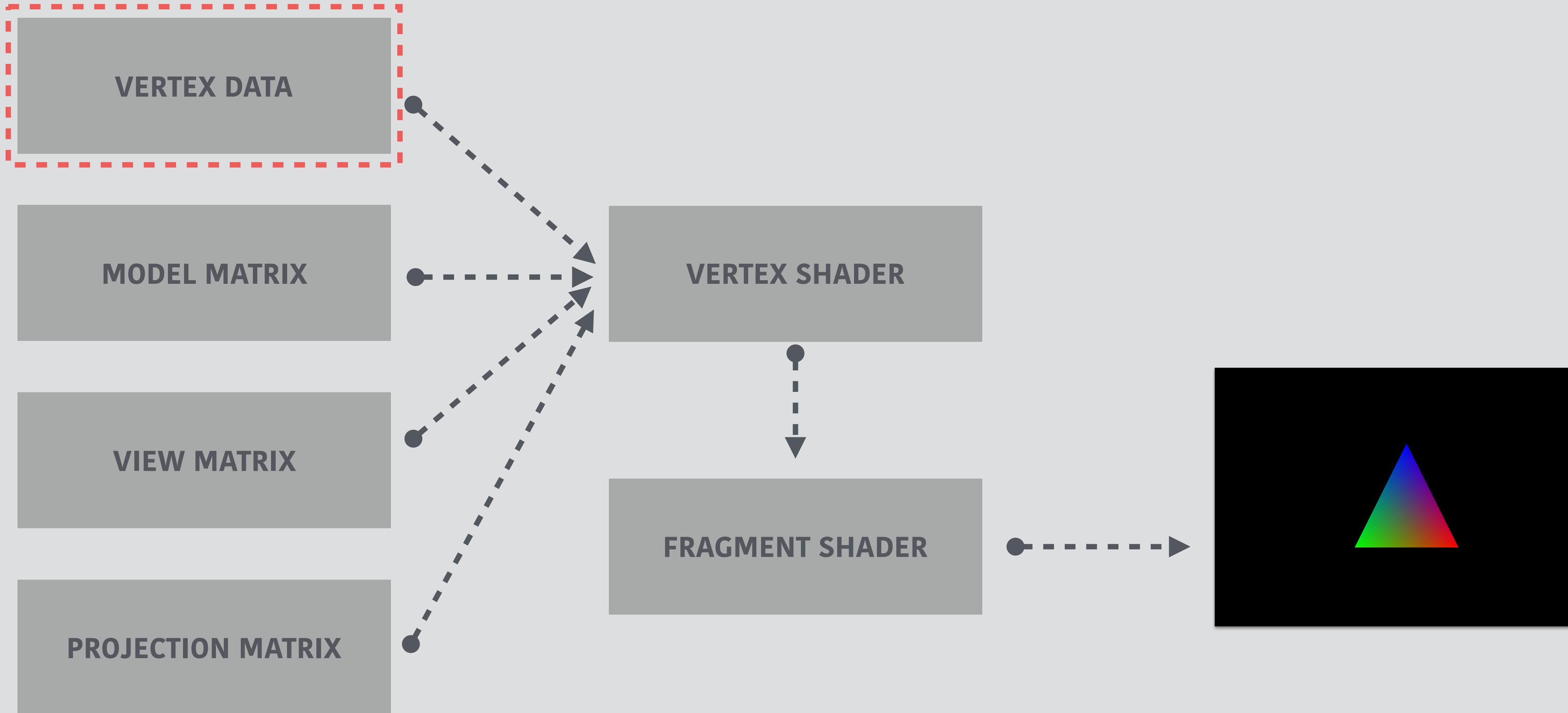


**Drawing polygons.
(happens every frame!)**

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);
```

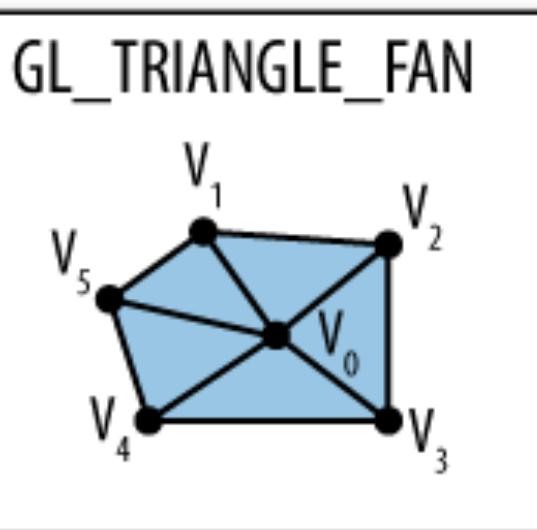
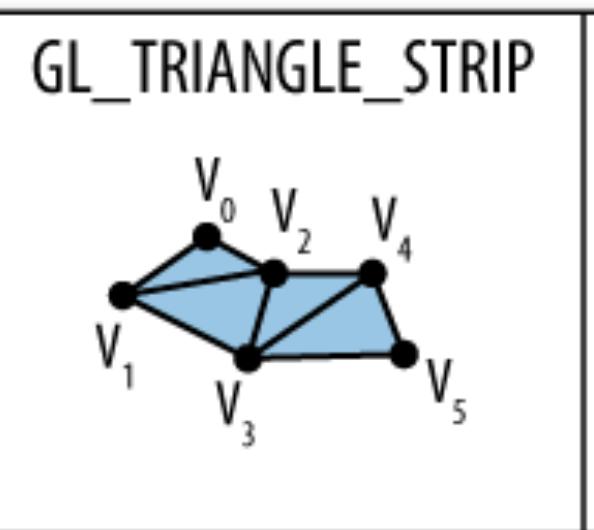
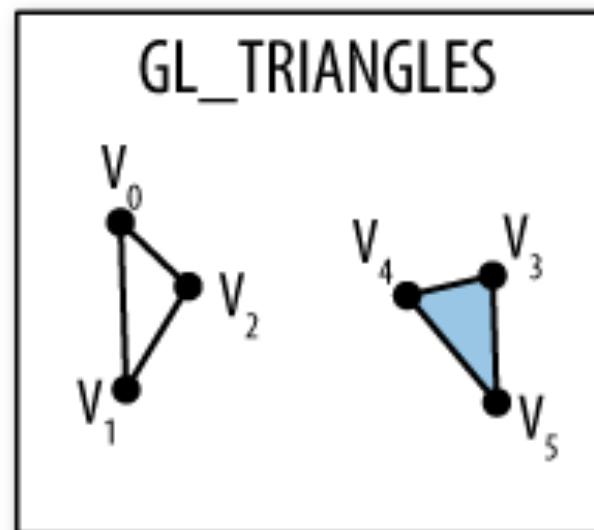
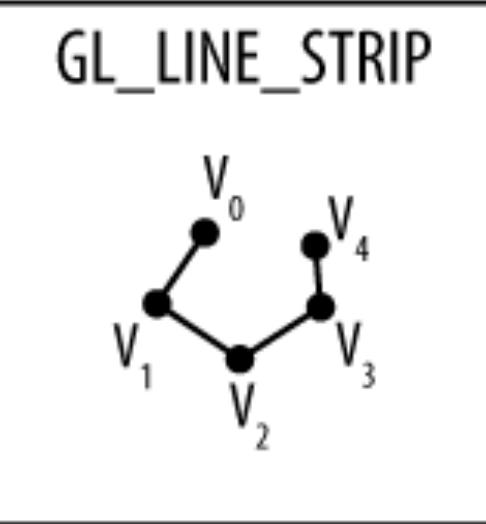
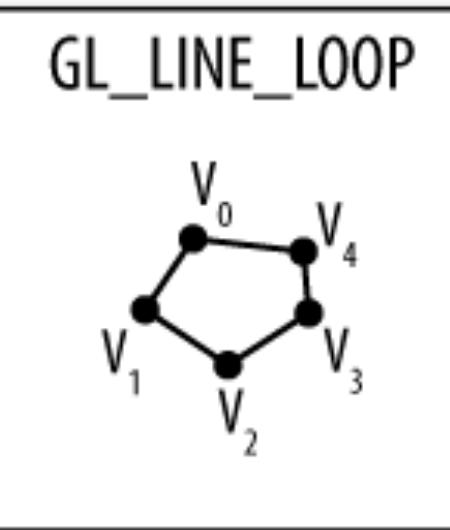
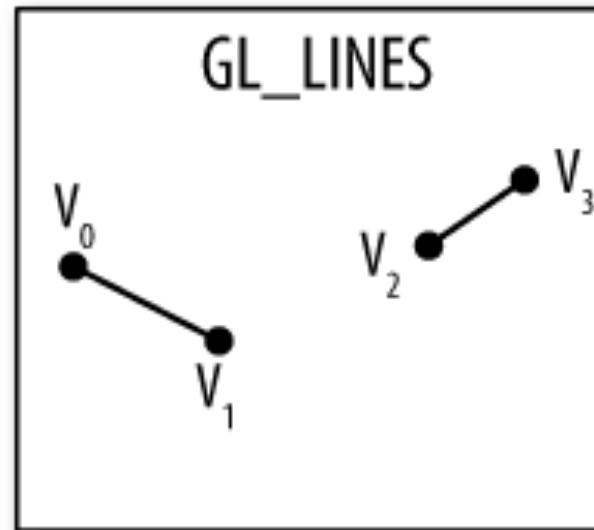
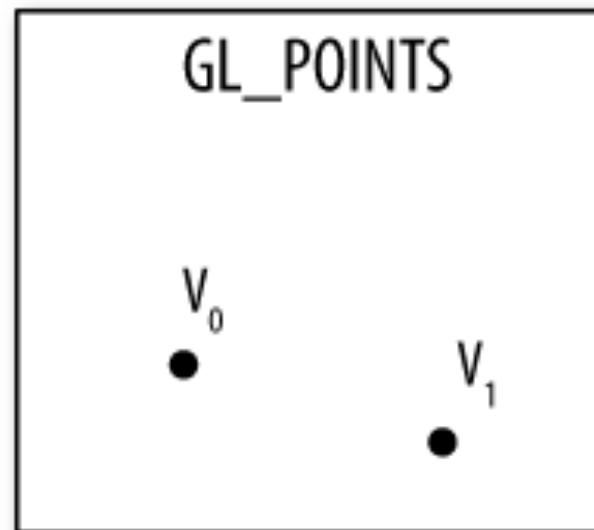
Enables a **vertex attribute**.

```
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);

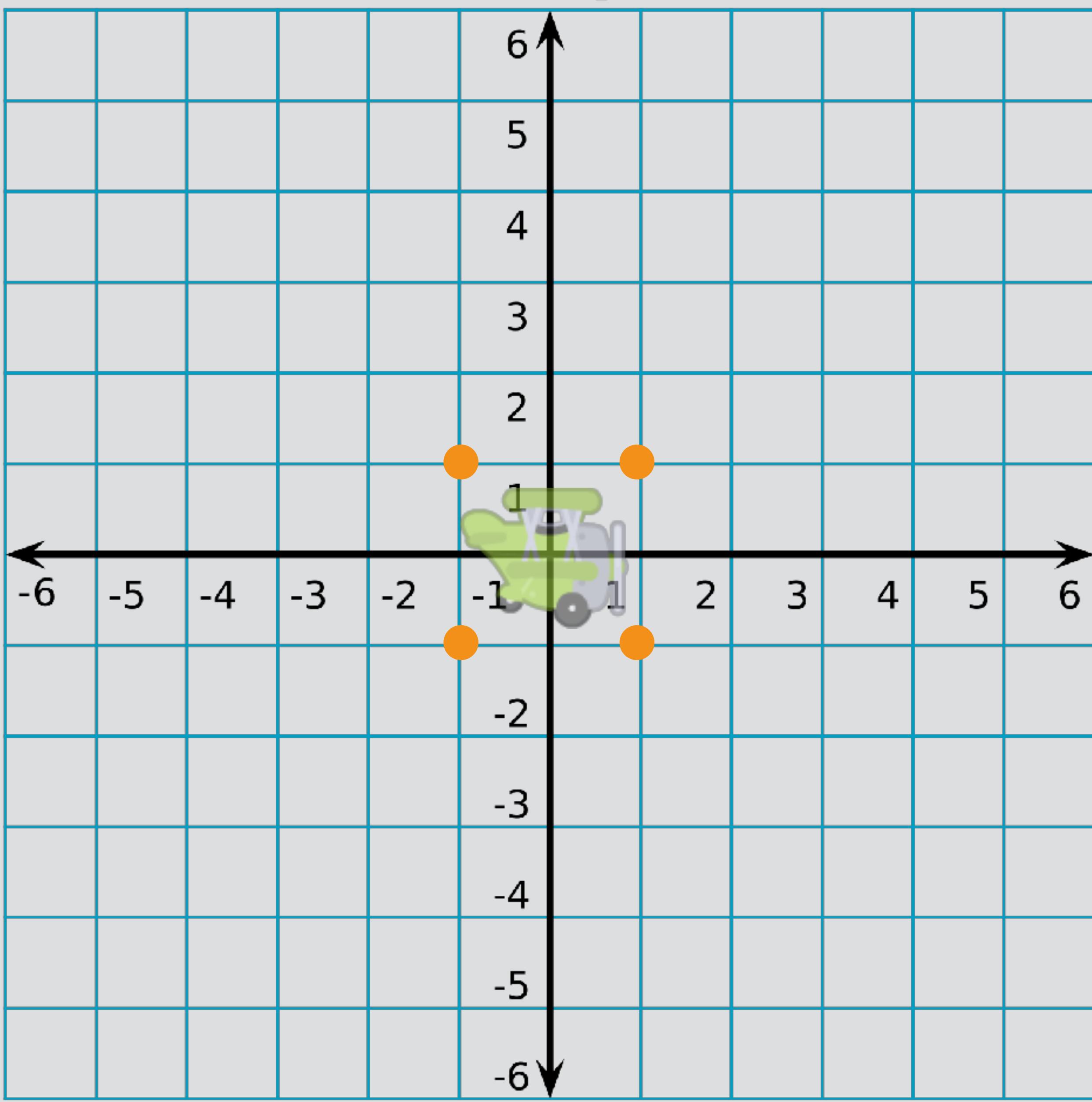
SDL_GL_SwapWindow(displayWindow);
```

Transformations

Identity matrix.

y-axis

x-axis



```
glm::mat4(1.0f)
```

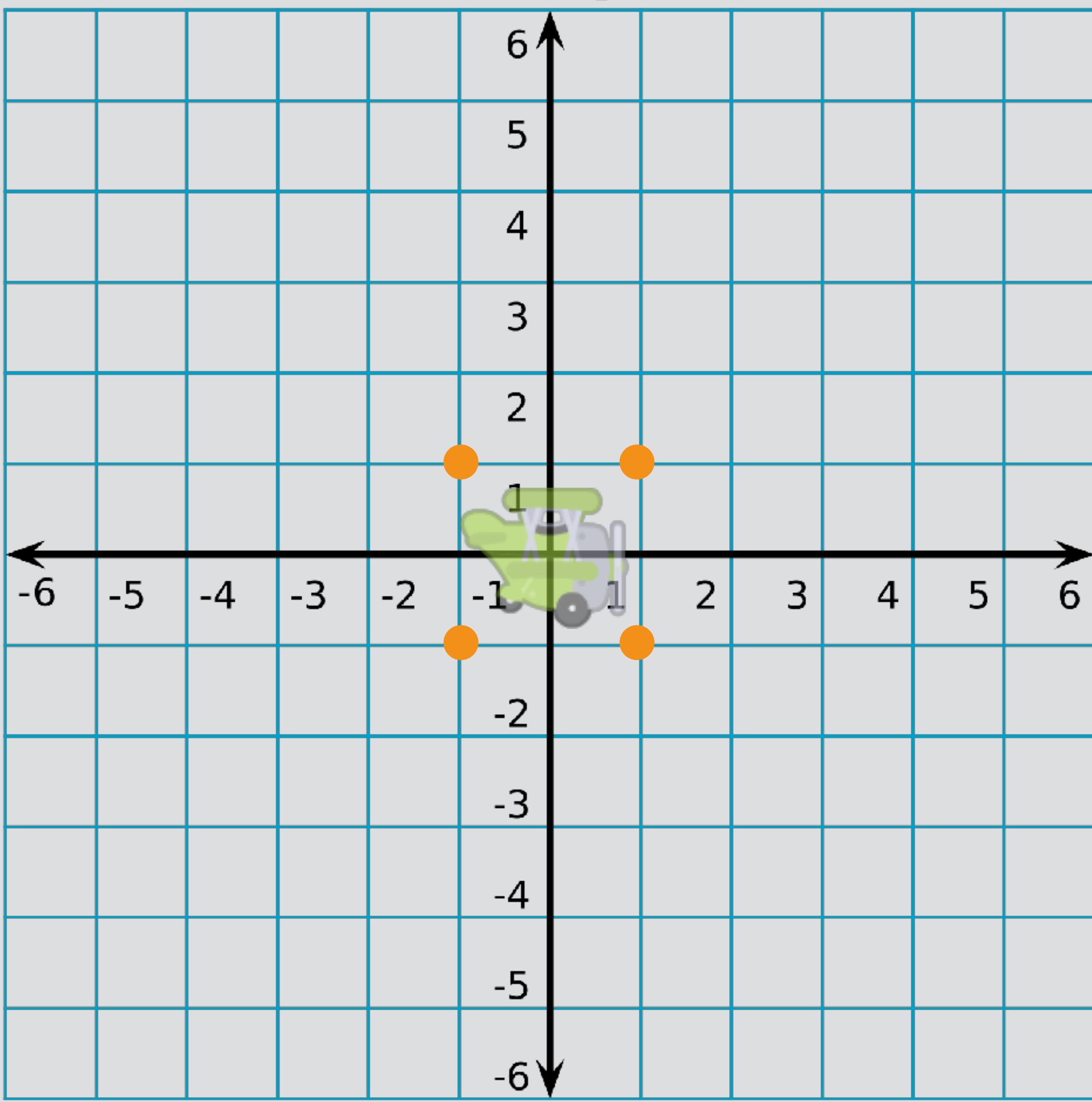
Creates an identity matrix.

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

Translations

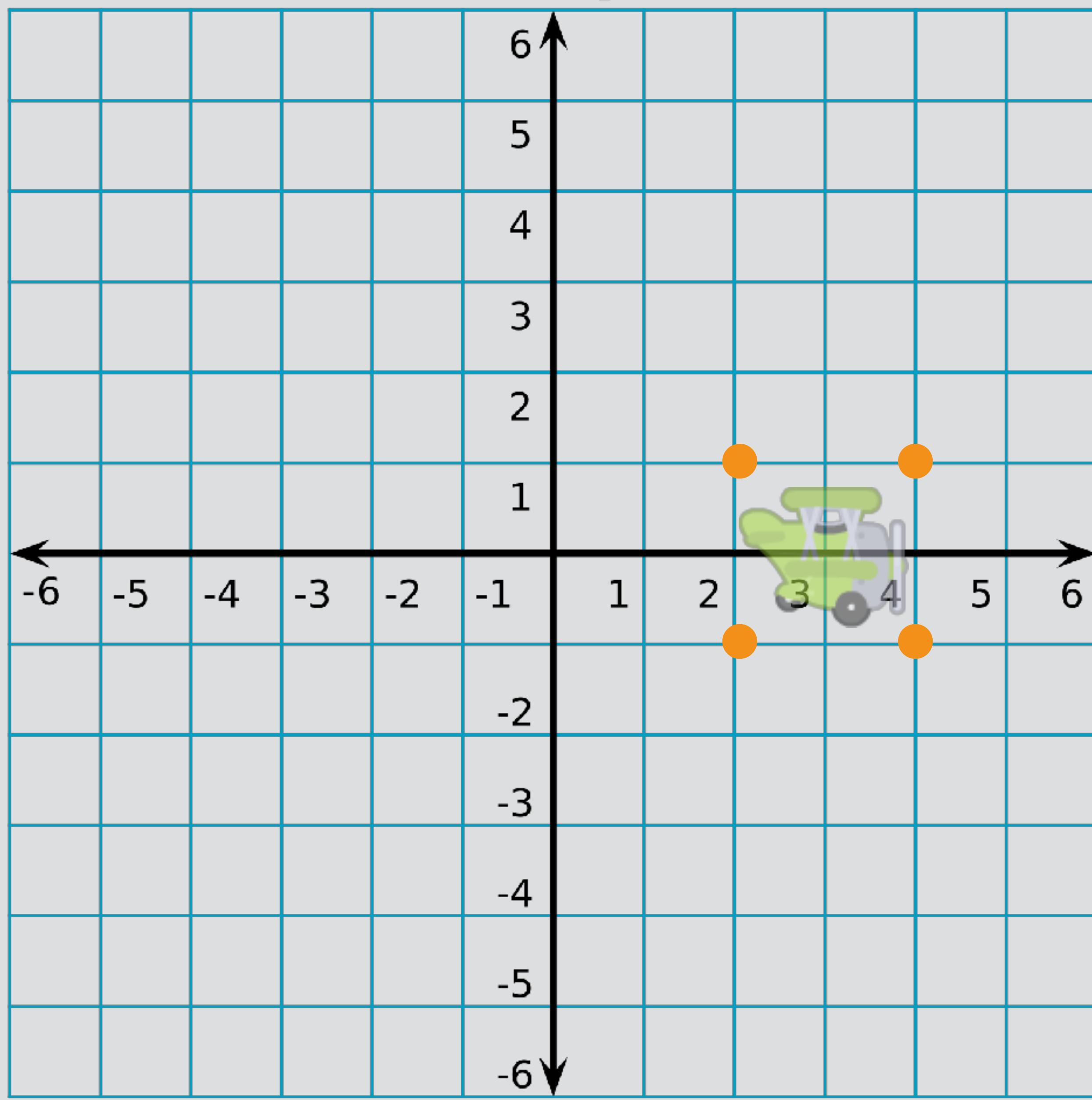
y-axis

x-axis



y-axis

x-axis



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

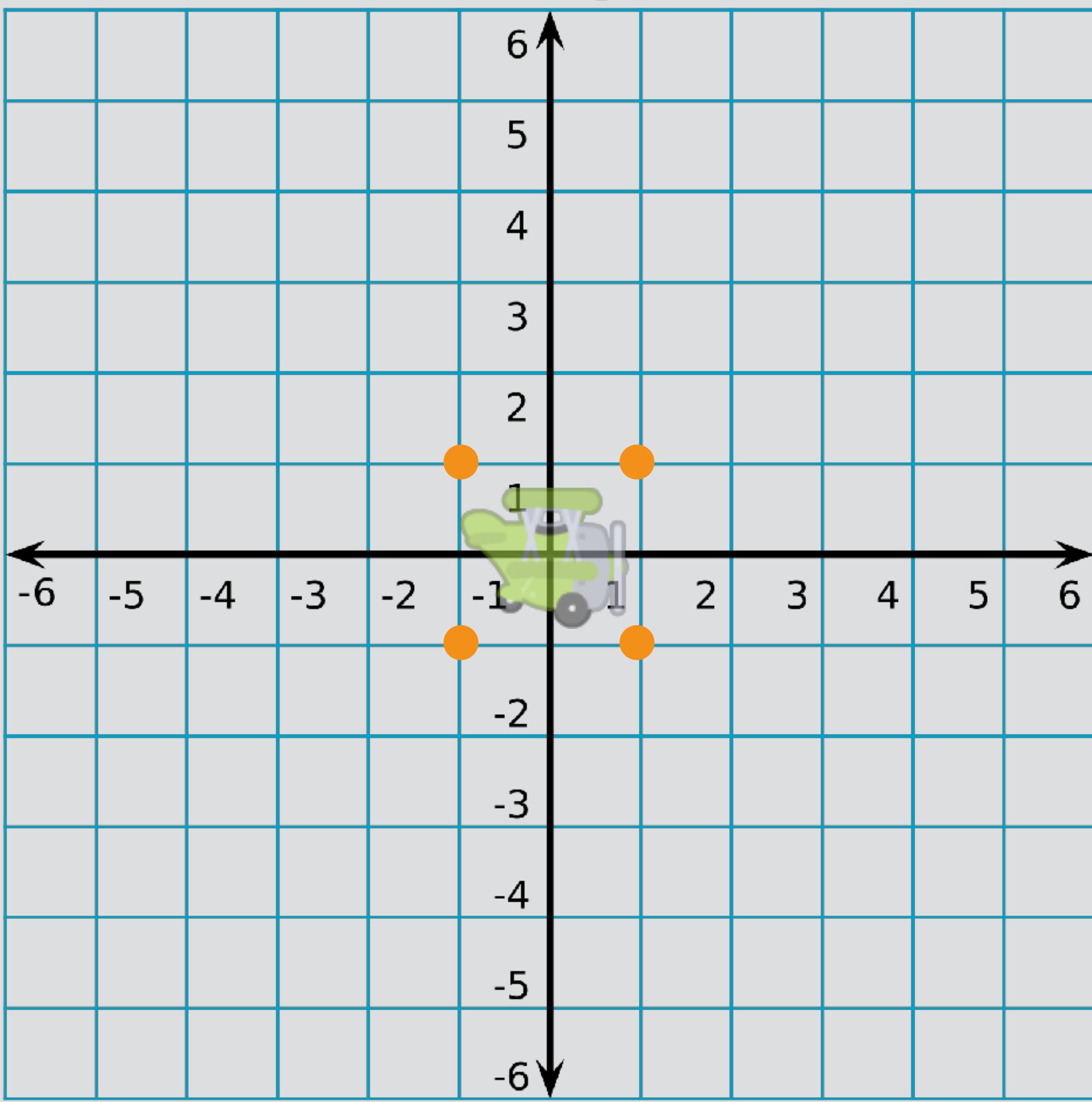
Translates a matrix by specified coordinates.

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

Scaling

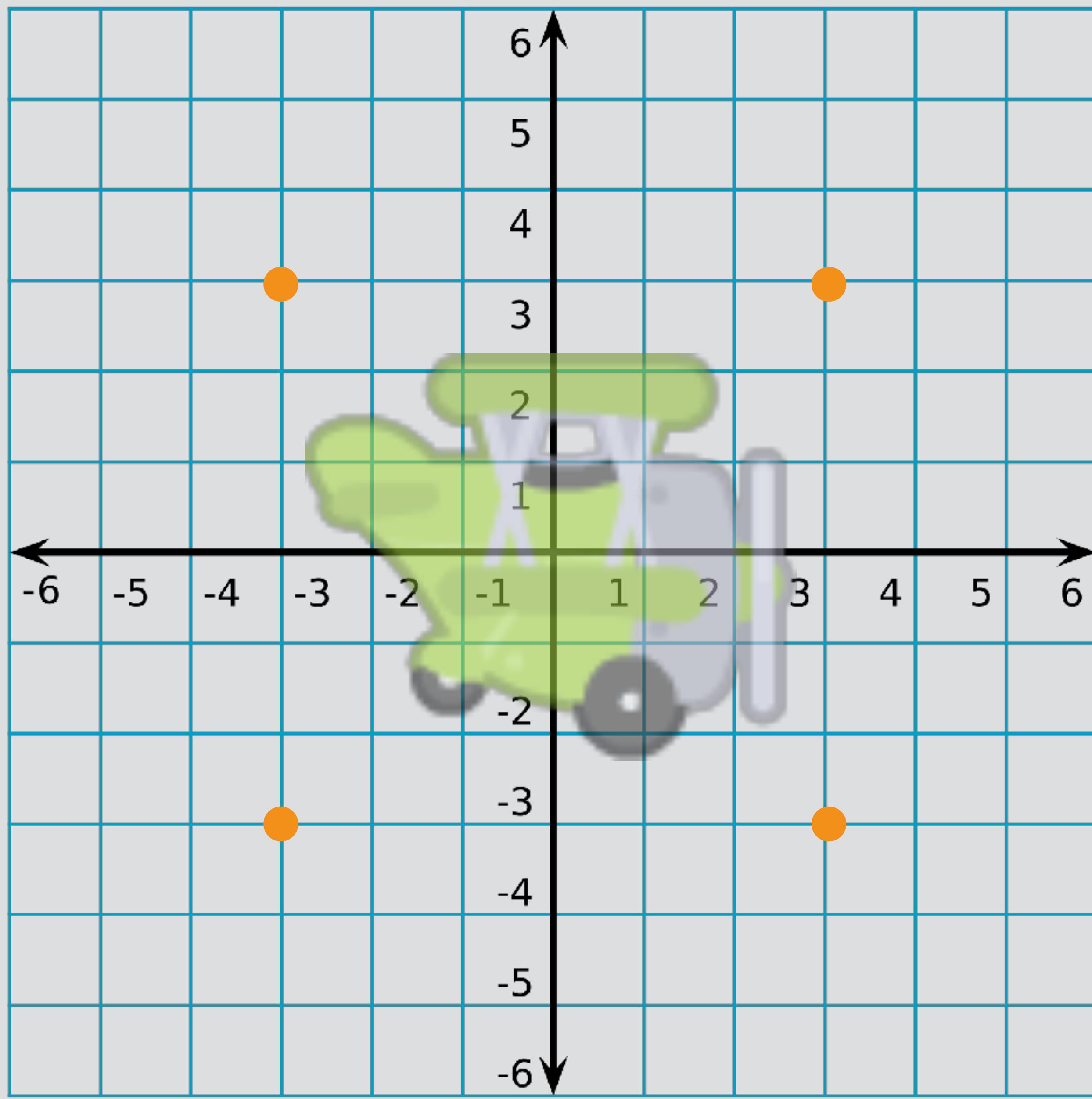
y-axis

x-axis



y-axis

x-axis



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

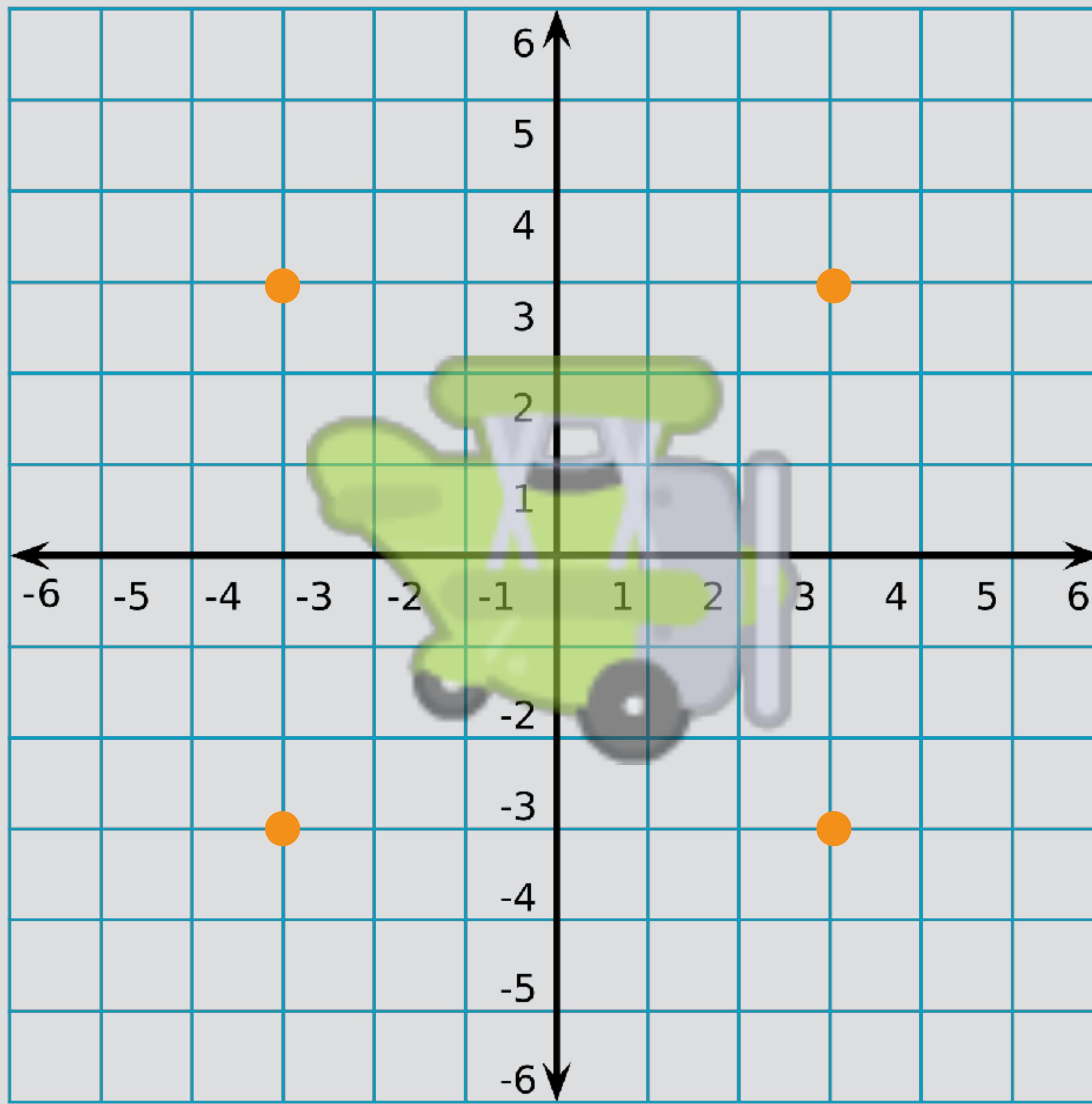
Scales a matrix by specified coordinates.

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

Rotation

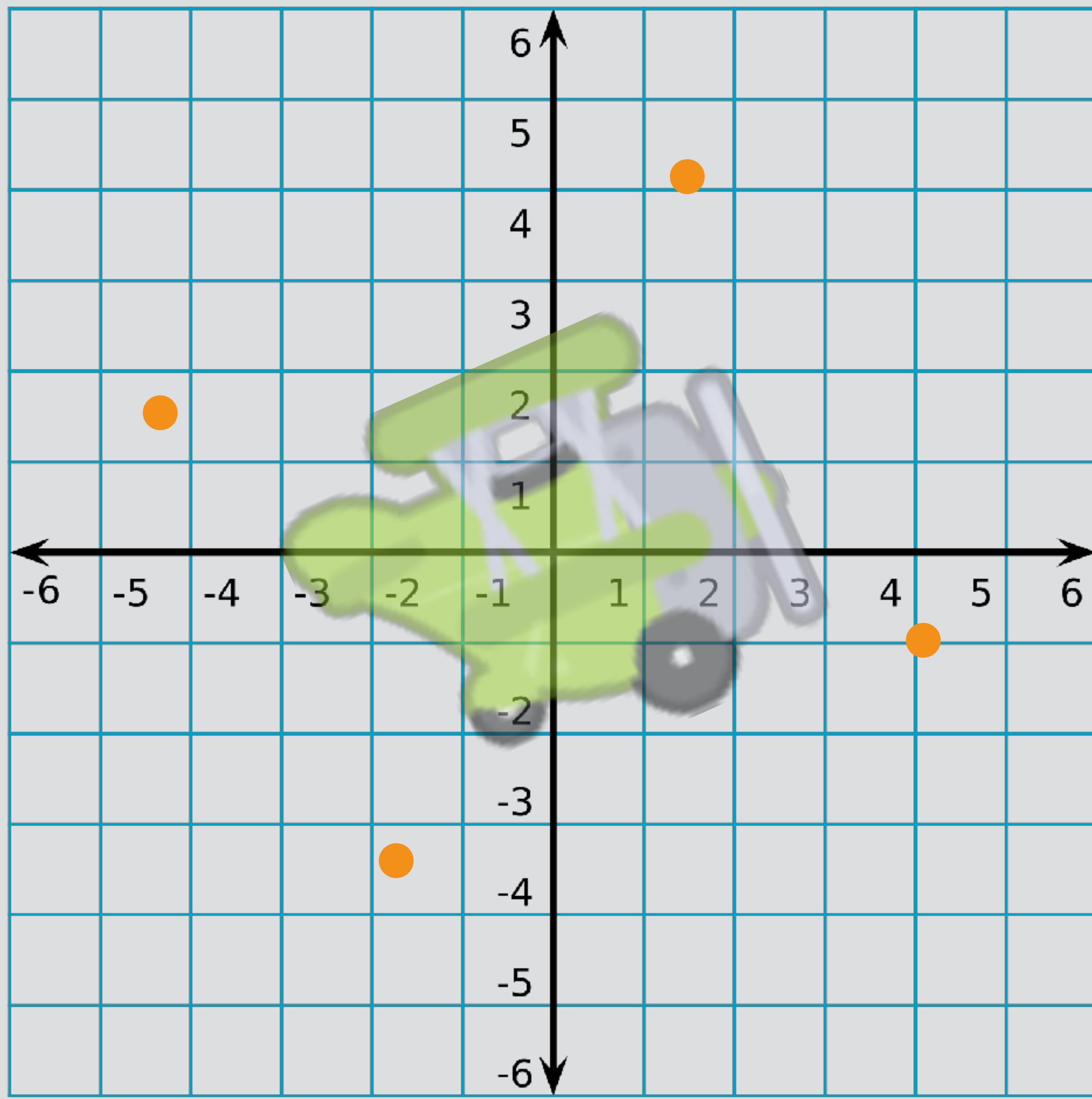
y-axis

x-axis



y-axis

x-axis



```
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**:

$$\text{RADIAN} = \text{DEGREE} * (\text{PI} / 180)$$

**Matrix transformations
are not commutative!**

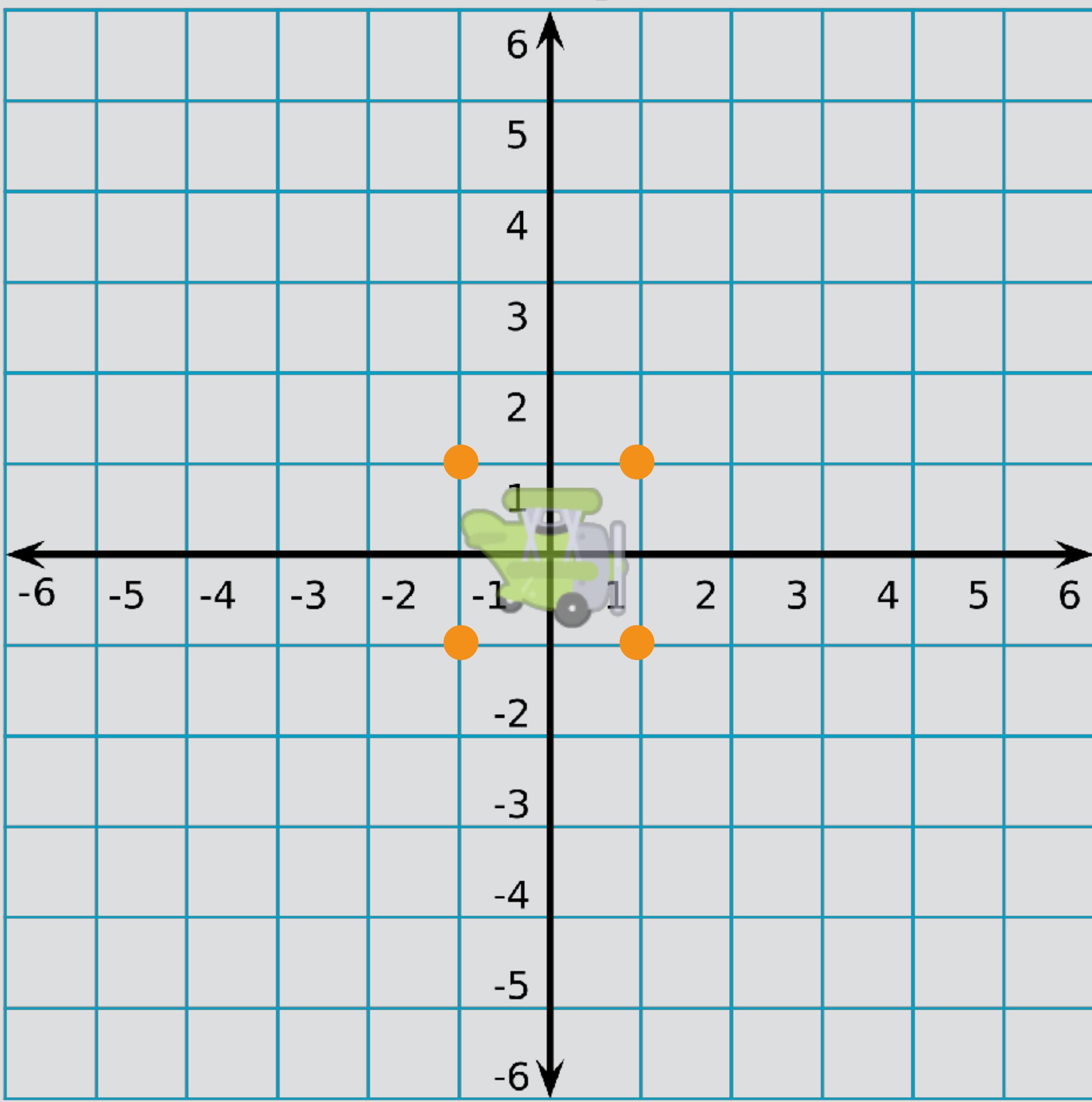
```
glm::mat4 modelMatrix = glm::mat4(1.0f);

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));
```

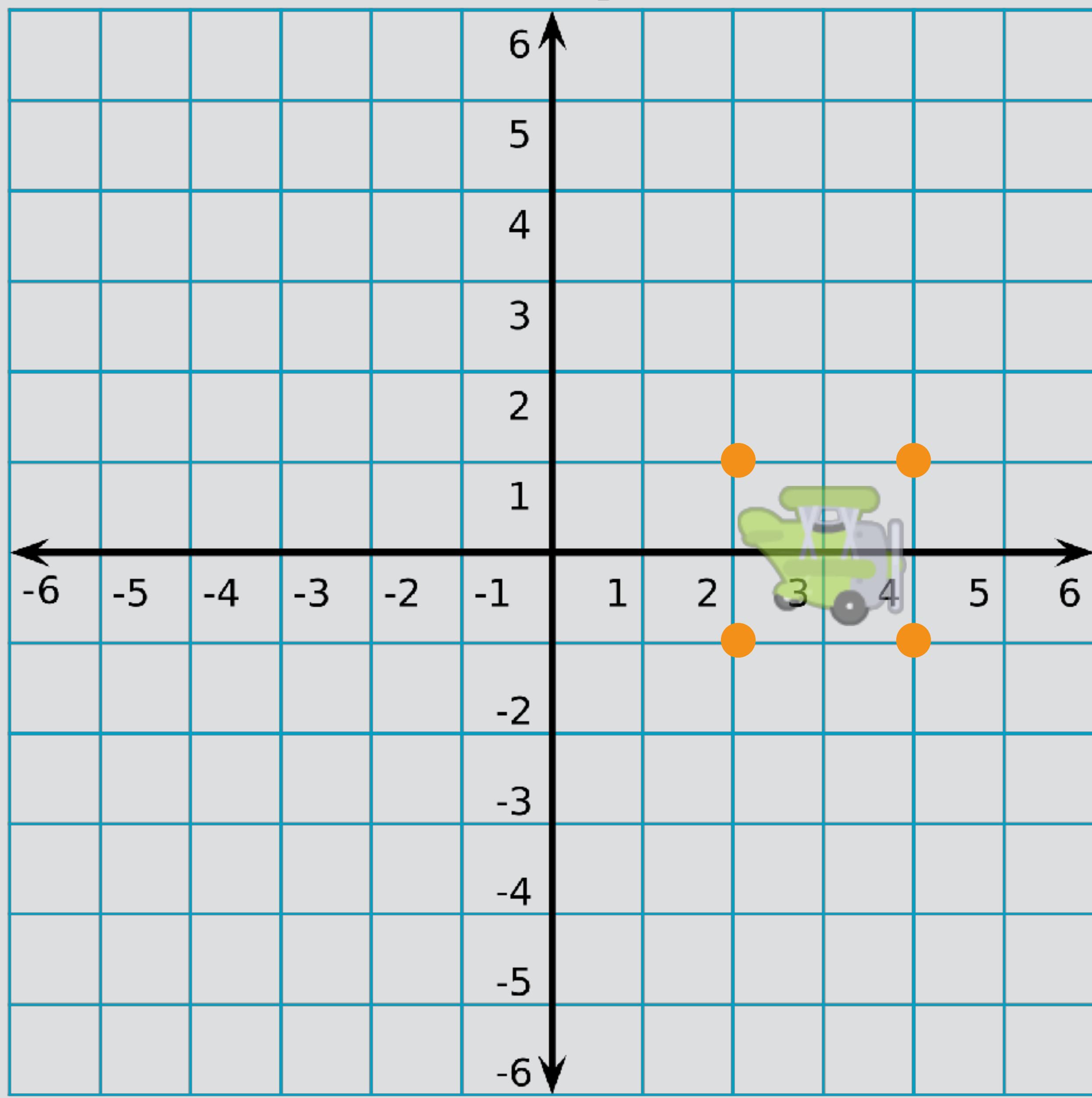
y-axis

x-axis



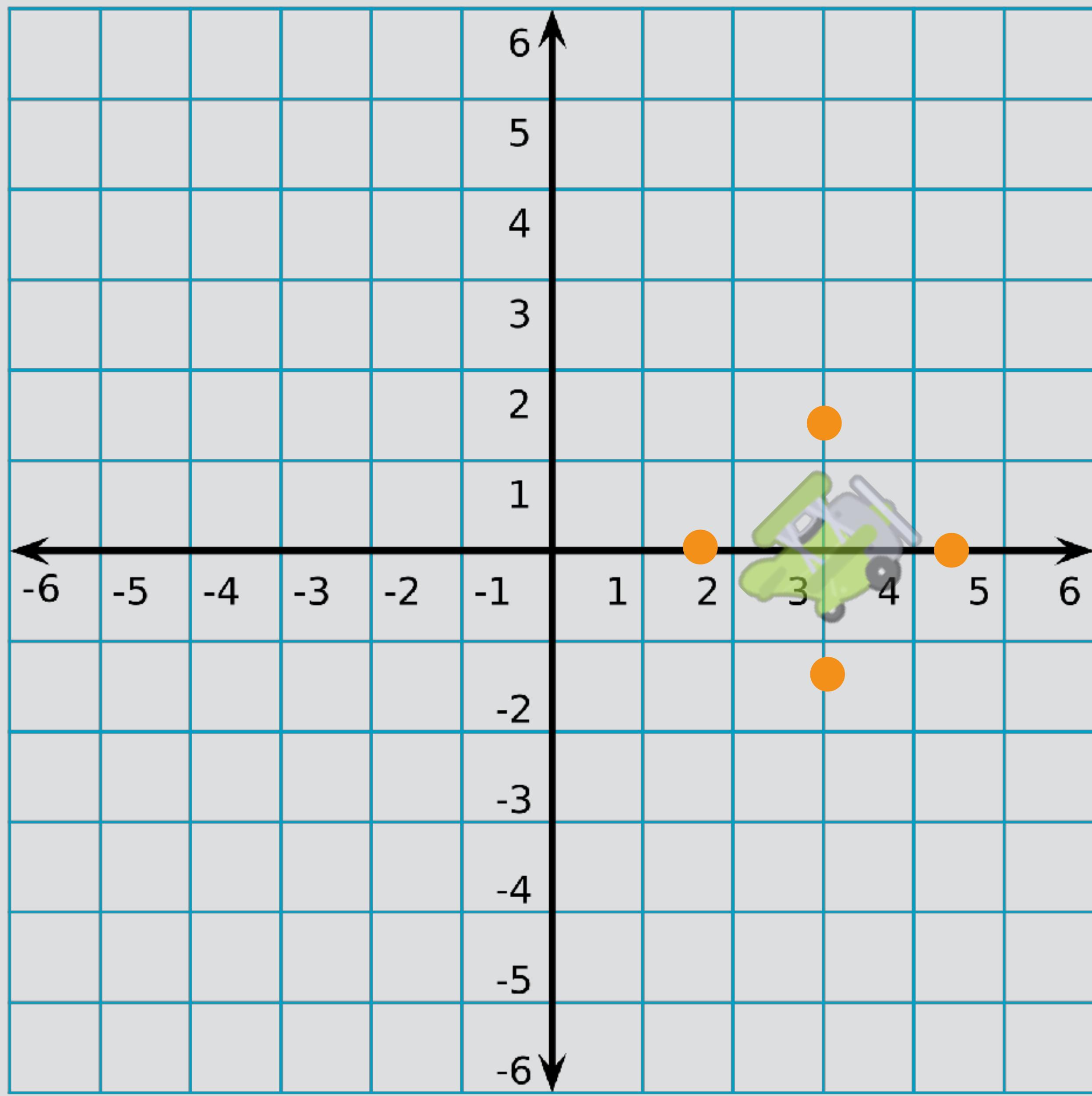
y-axis

x-axis



y-axis

x-axis



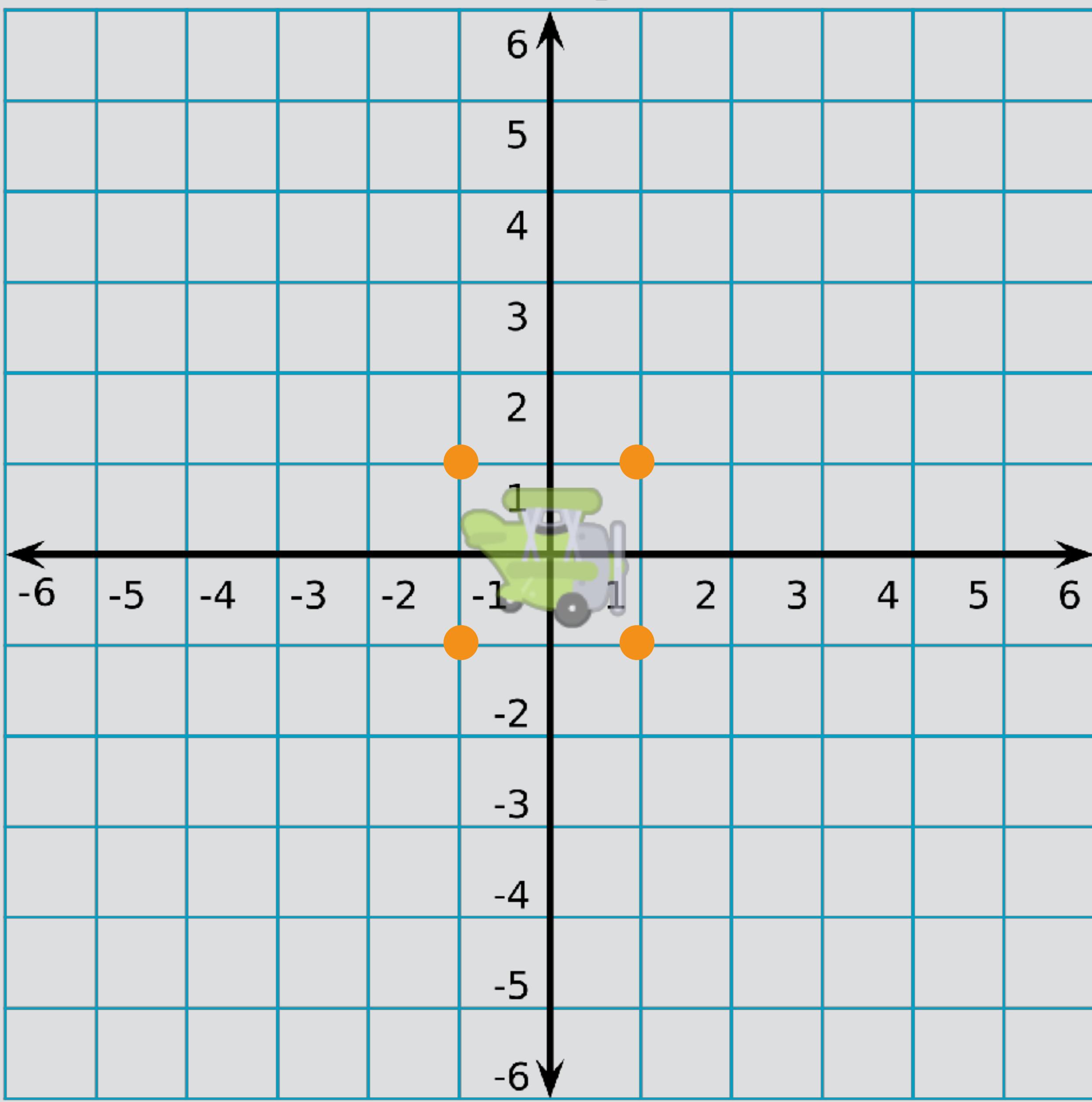
```
glm::mat4 modelMatrix = glm::mat4(1.0f);

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));
```

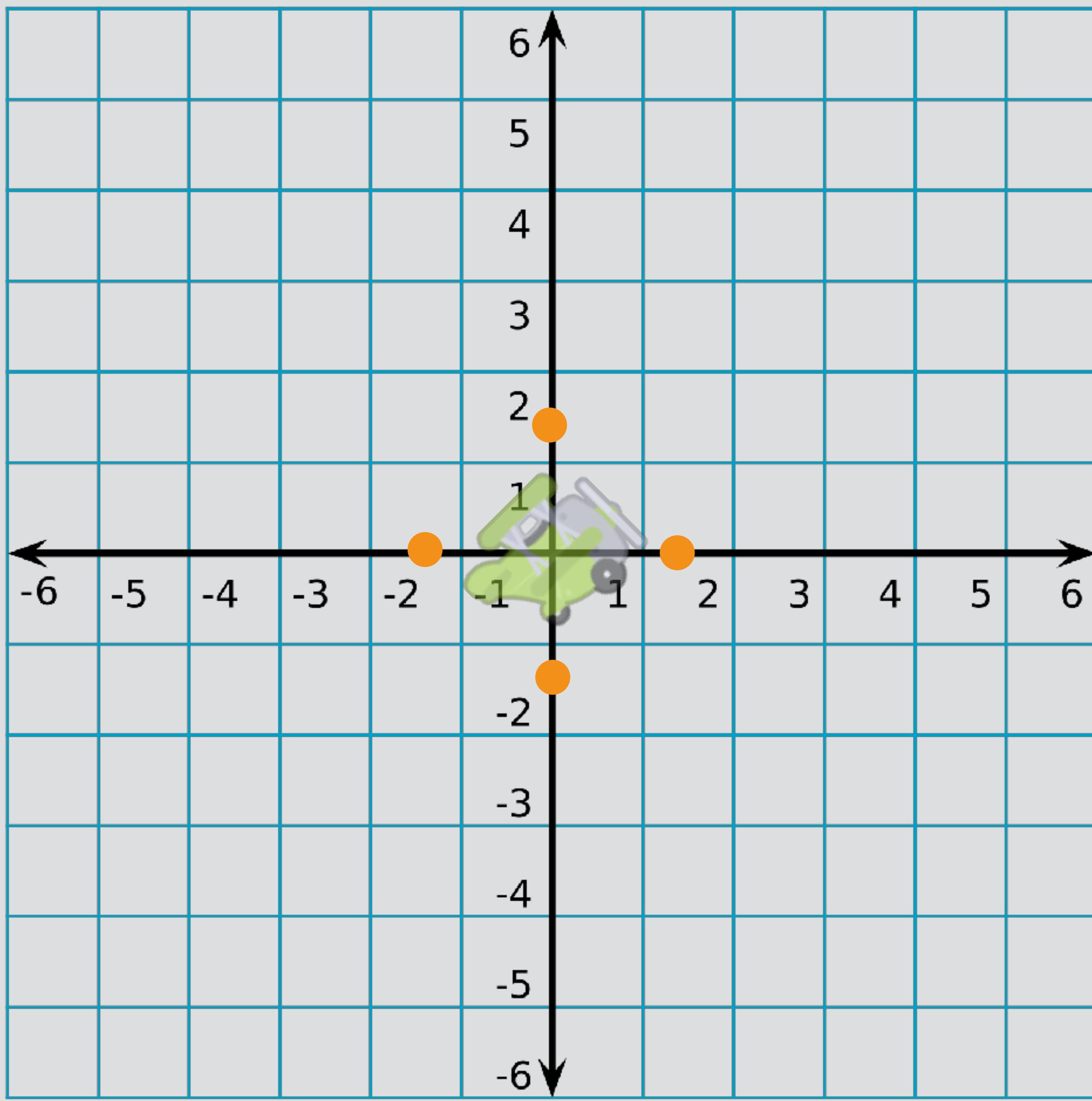
y-axis

x-axis



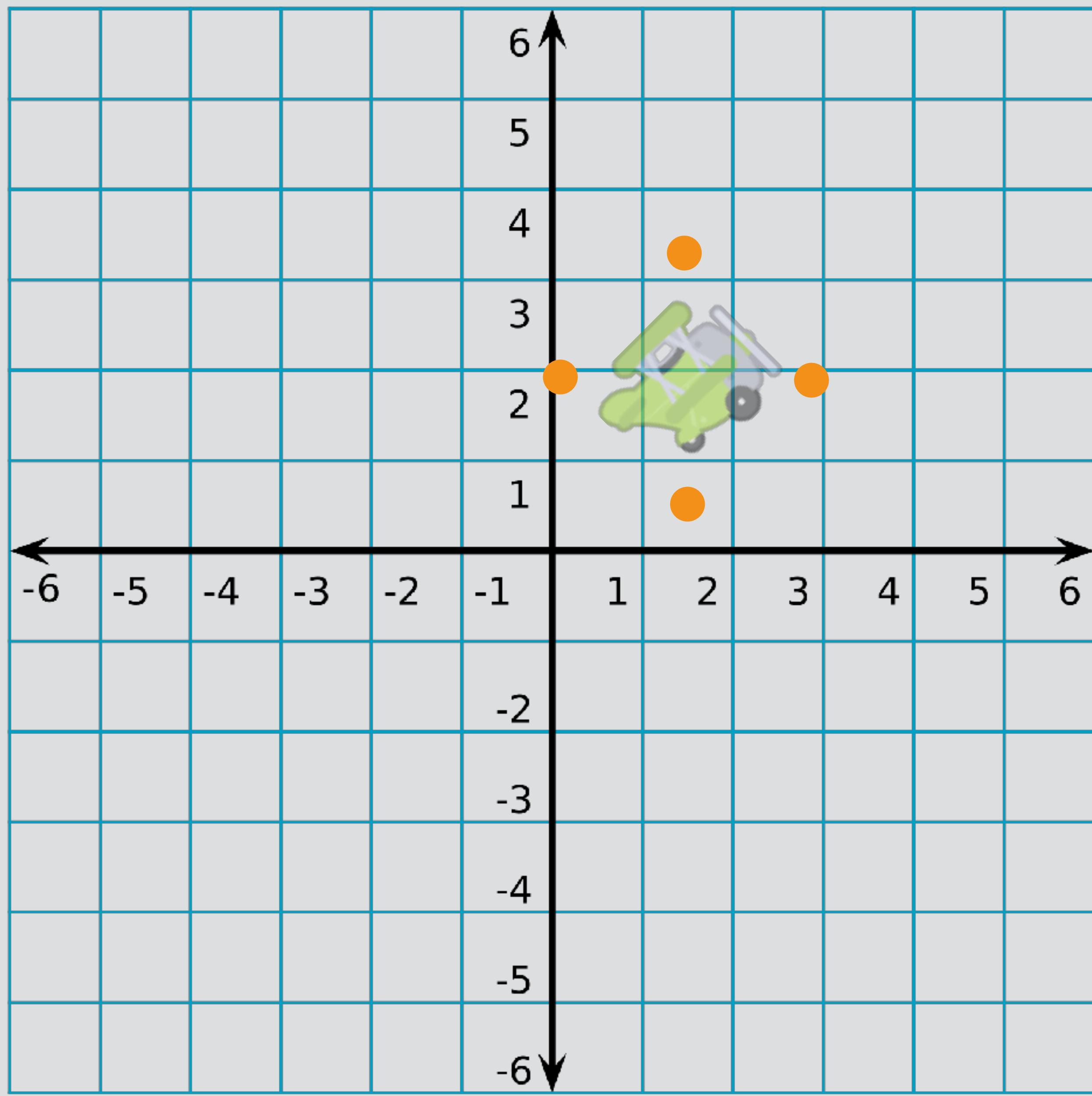
y-axis

x-axis



y-axis

x-axis



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