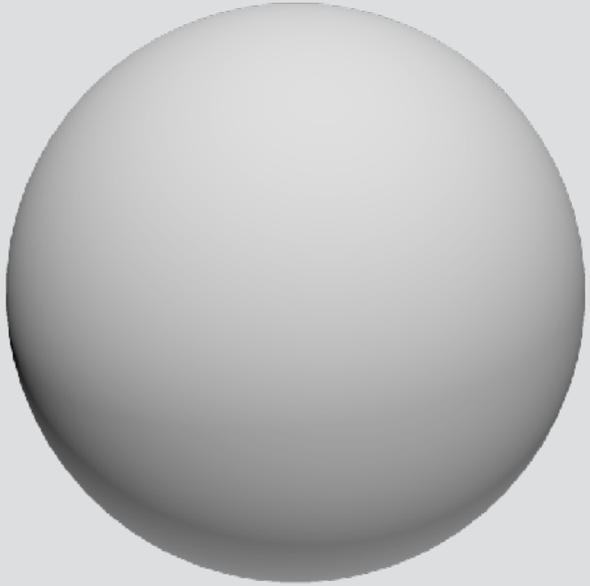
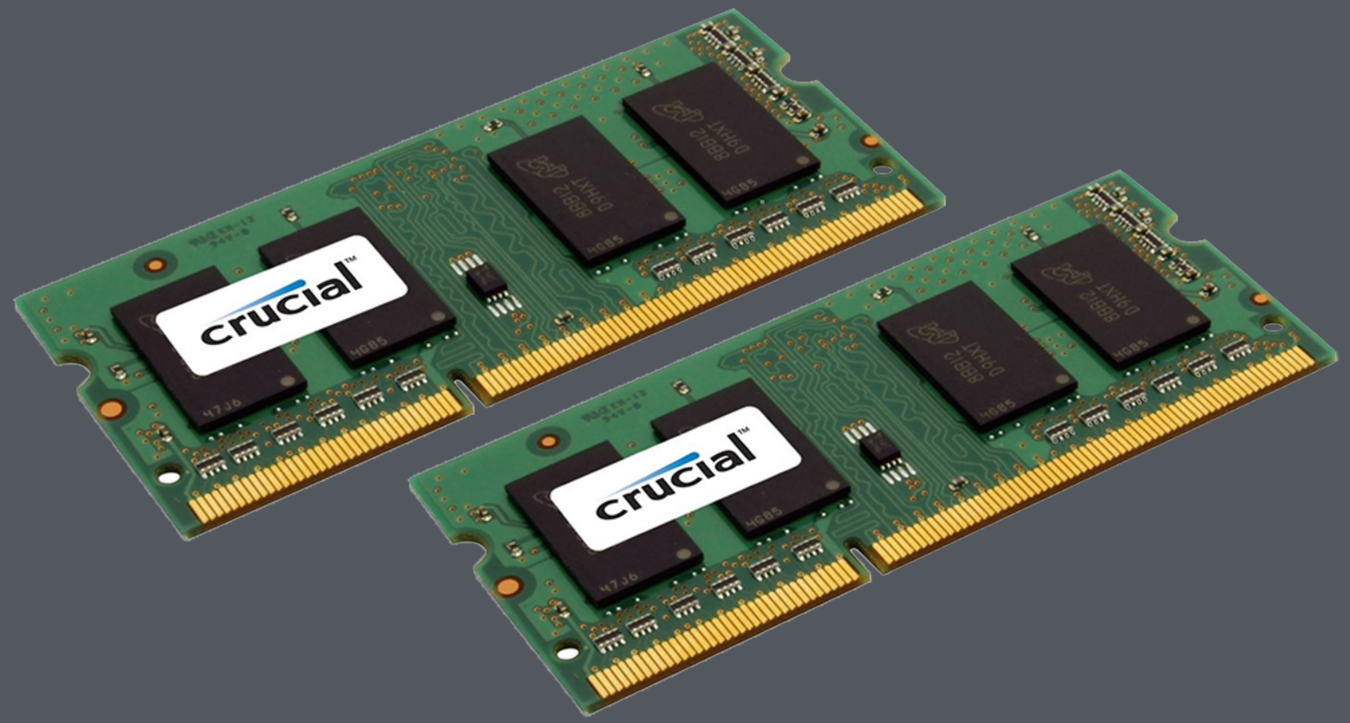
Graphics Foundations Part 1



CS 3113

Quick pointer refresher



1 0 0 1
0 1 0 1

0 1 2 3 4 5 6 7 8 9 ... 7999999996 7999999997 7999999998 7999999999

```
char myVar = ‘a’; // char is an 8-bit variable
```


```
float myVar = 1.0f; // float is a 32-bit variable
```


```
double myVar = 1.0; // double is a 64-bit variable
```

```
float myVar = 16.0f;
```


```
float *myVarPtr = &myVar;

cout << myVarPtr << endl; // prints 3
```

```
float myVar = 16.0f;

float *myVarPtr = &myVar;

cout << myVarPtr << endl; // prints 3

cout << *myVarPtr << endl; // prints 16

cout << &myVar << endl; // prints 3
```

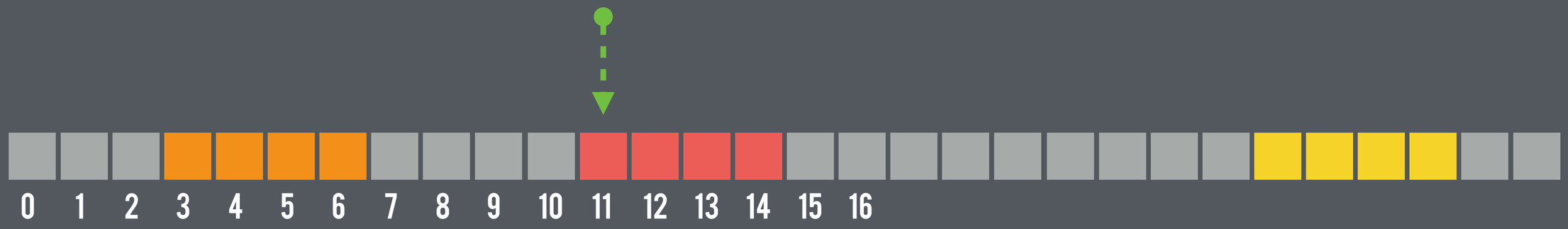
Reference

Dereference

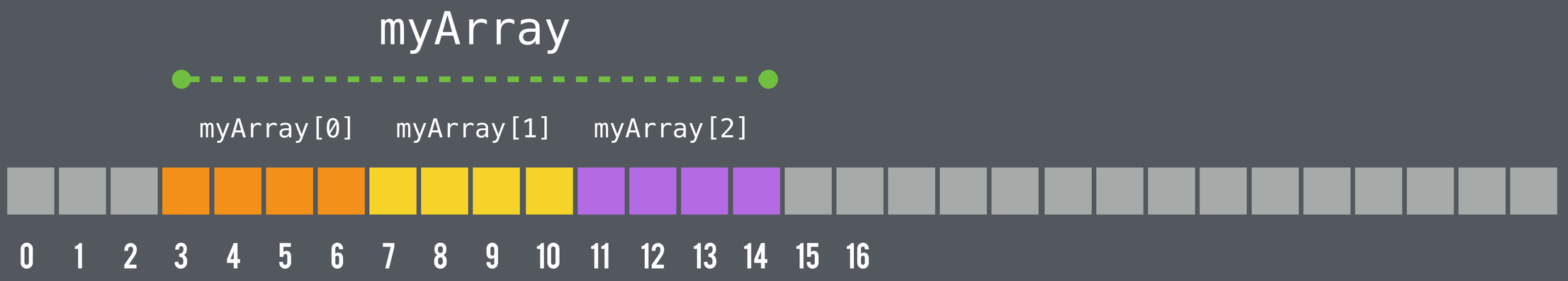
```
float myVar = 16.0f;

float *myVarPtr = &myVar;

cout << myVarPtr; // prints 3

cout << *myVarPtr; // prints 16

cout << &myVar; // prints 3

float **myVarPtrPtr = &myVarPtr;

cout << myVarPtrPtr; // prints 11
```

Reference

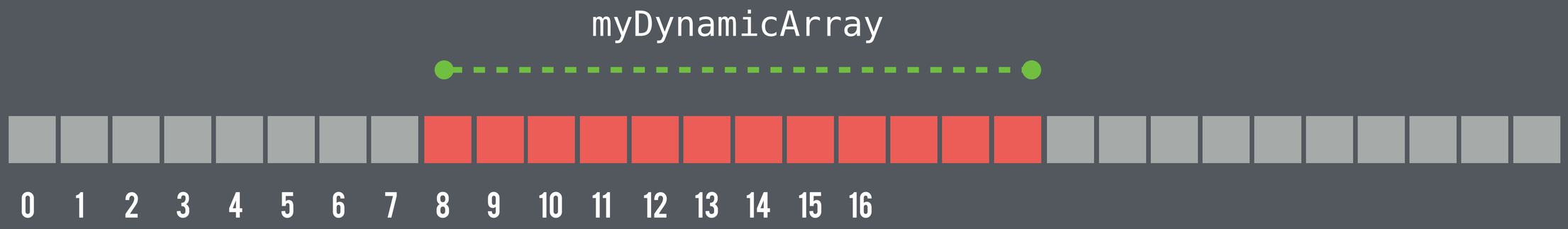
Dereference



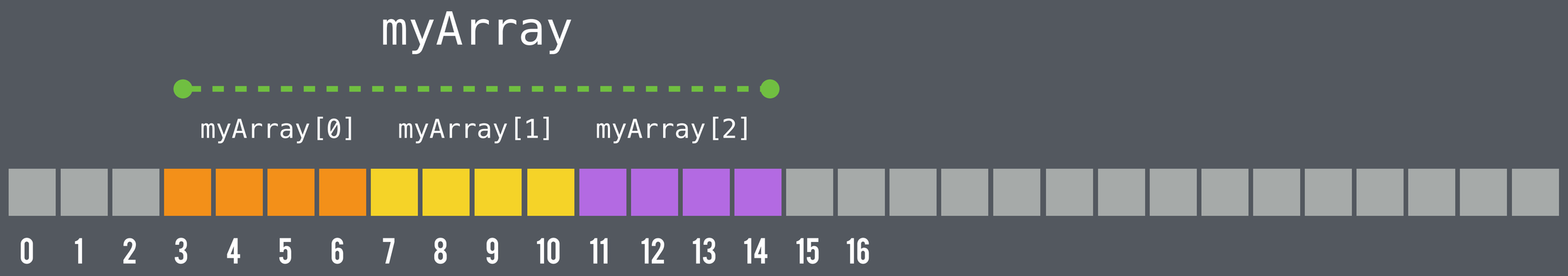
```
float myArray[3] = {13.5f, 2.3f, 5.4f};

cout << myArray[0]; // prints 13.5
cout << myArray; // prints 3
cout << &myArray[0]; // prints 3

float *myArrayAsAPointer = myArray;
cout << myArrayAsAPointer; // prints 3

cout << myArrayAsAPointer[0]; // prints 13.5
cout << myArrayAsAPointer[1]; // prints 2.3
```



```
float *myDynamicArray;

cout << myDynamicArray; // 0x0 or some uninitialized location
cout << myDynamicArray[0]; // crash or garbage!

myDynamicArray = new float[3];

cout << myDynamicArray; // prints 8
cout << myDynamicArray[0]; no crash!

delete myDynamicArray;
```



```
float *myArray = new float[3];
```




```
char *myCharArray = (char*) myArray;
```

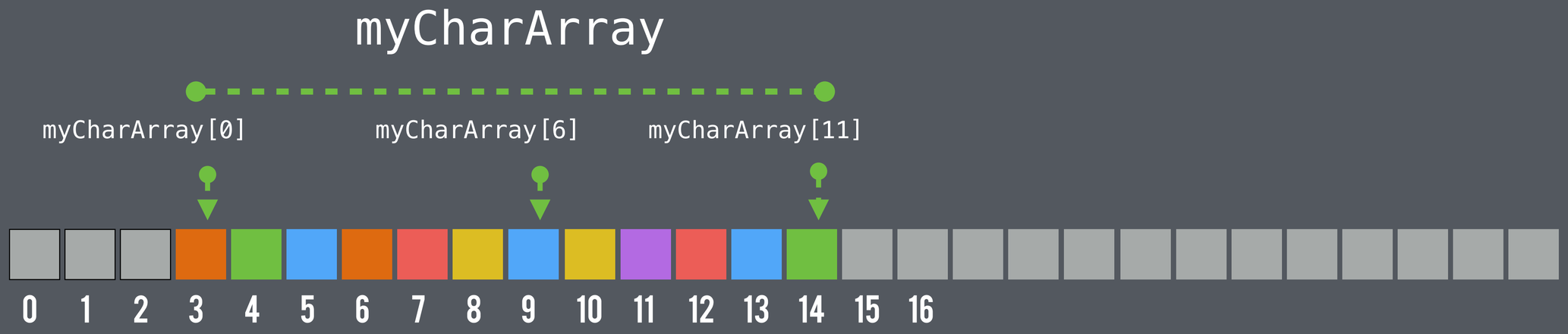


Image on the screen

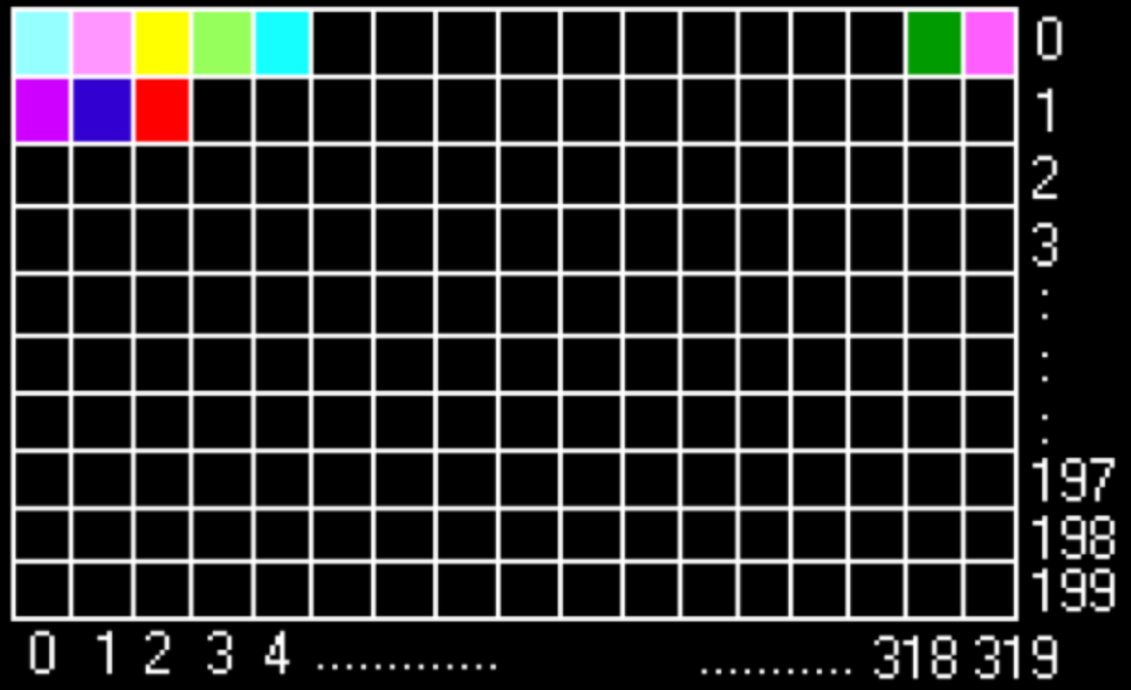
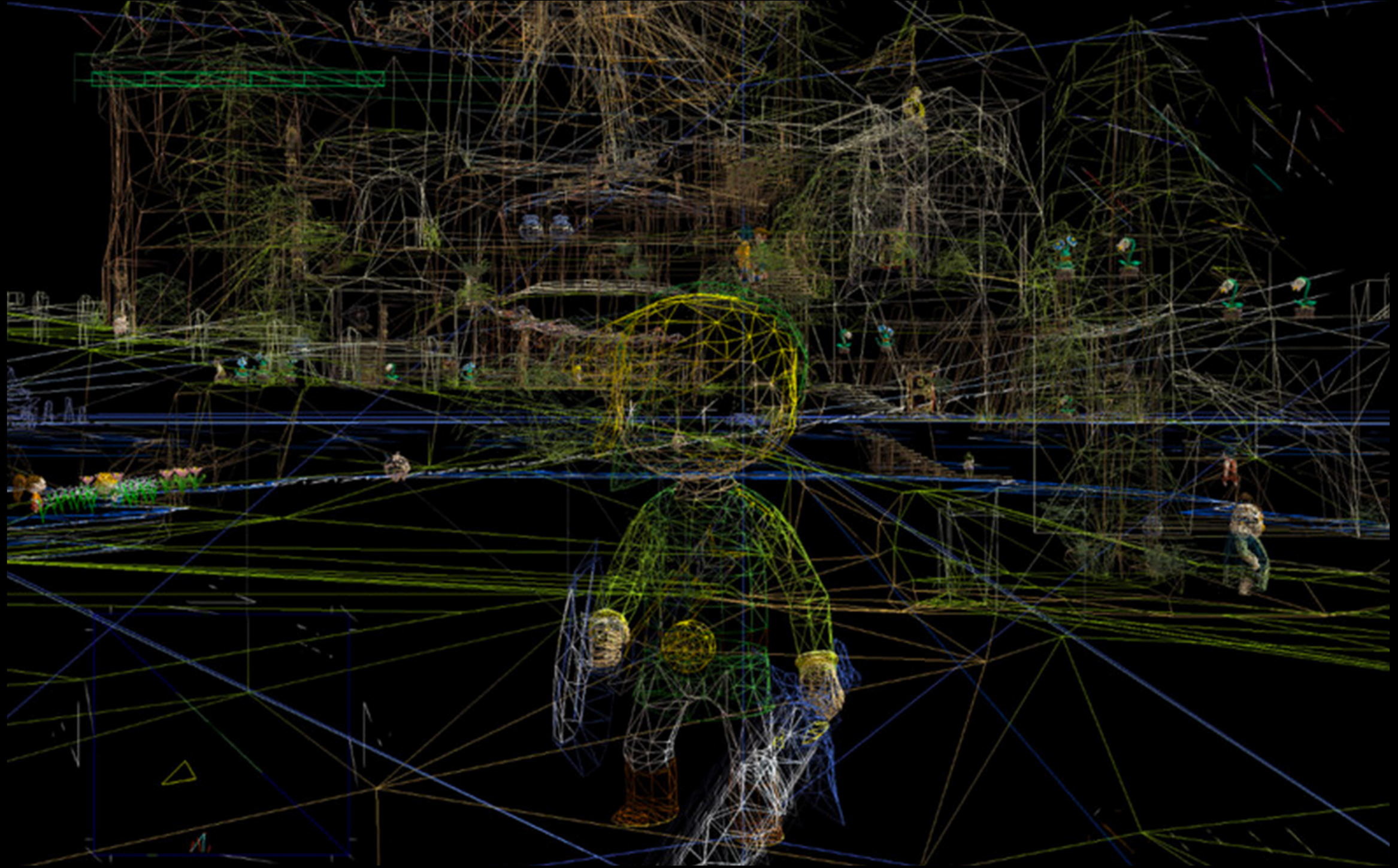
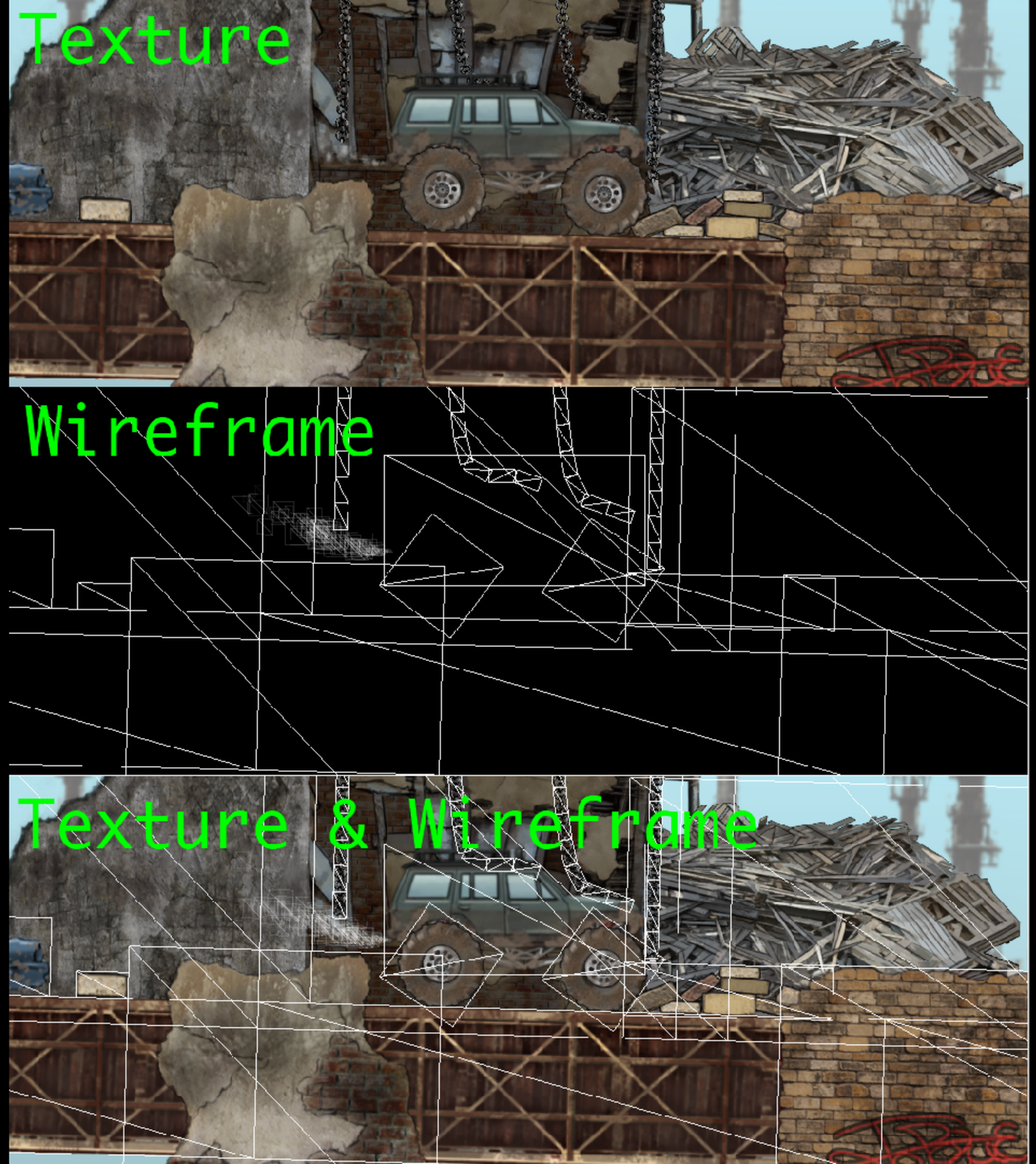


Image in memory

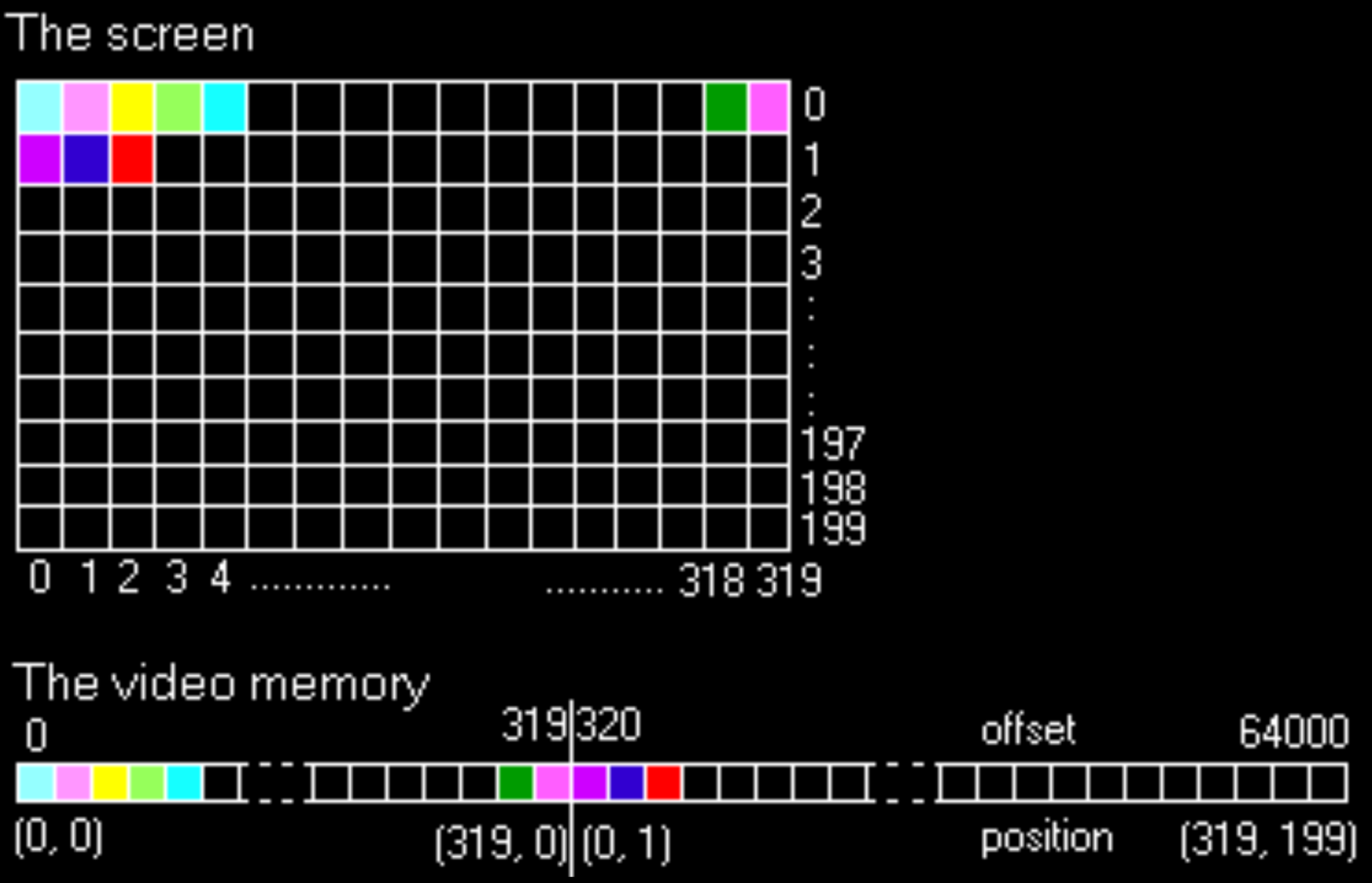
Graphics in games


Defend
Put Away
72
FREE
4168

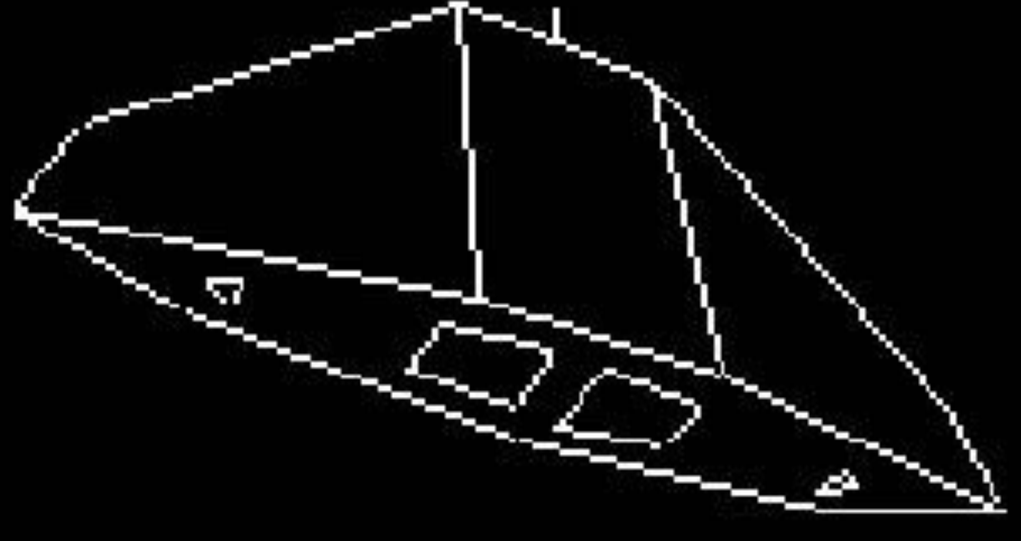



Texture
Wireframe
Texture & Wireframe

Why triangles?


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)?

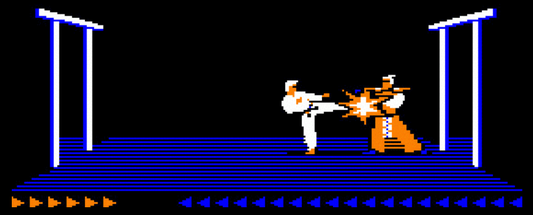

SPD: 0
VSI: 01.78
THR:90%
HDG:0
ALT:83

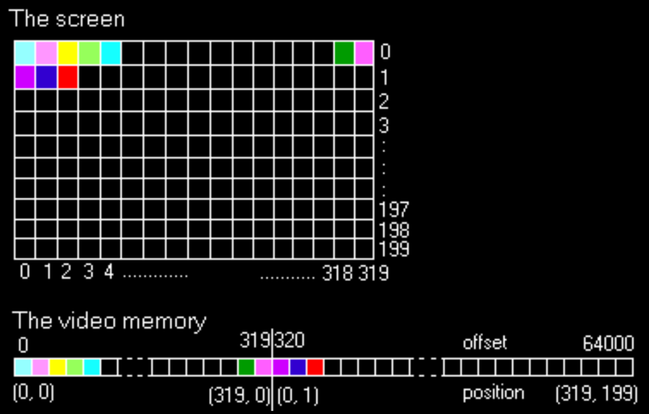

93
75
13
0
78
91
13


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

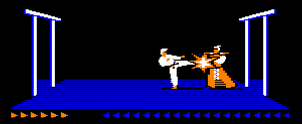
Software Rendering

RAM

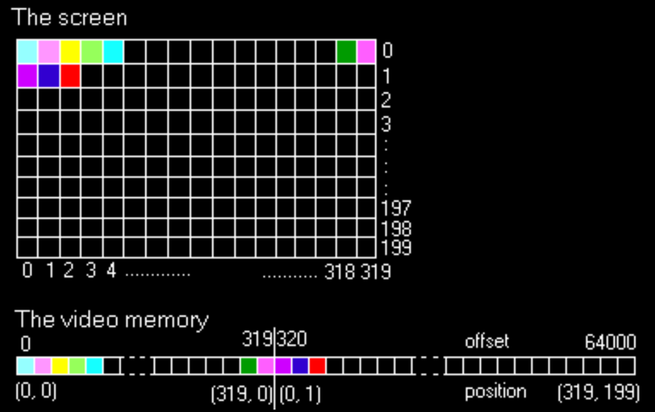




Video card



Software Rendering



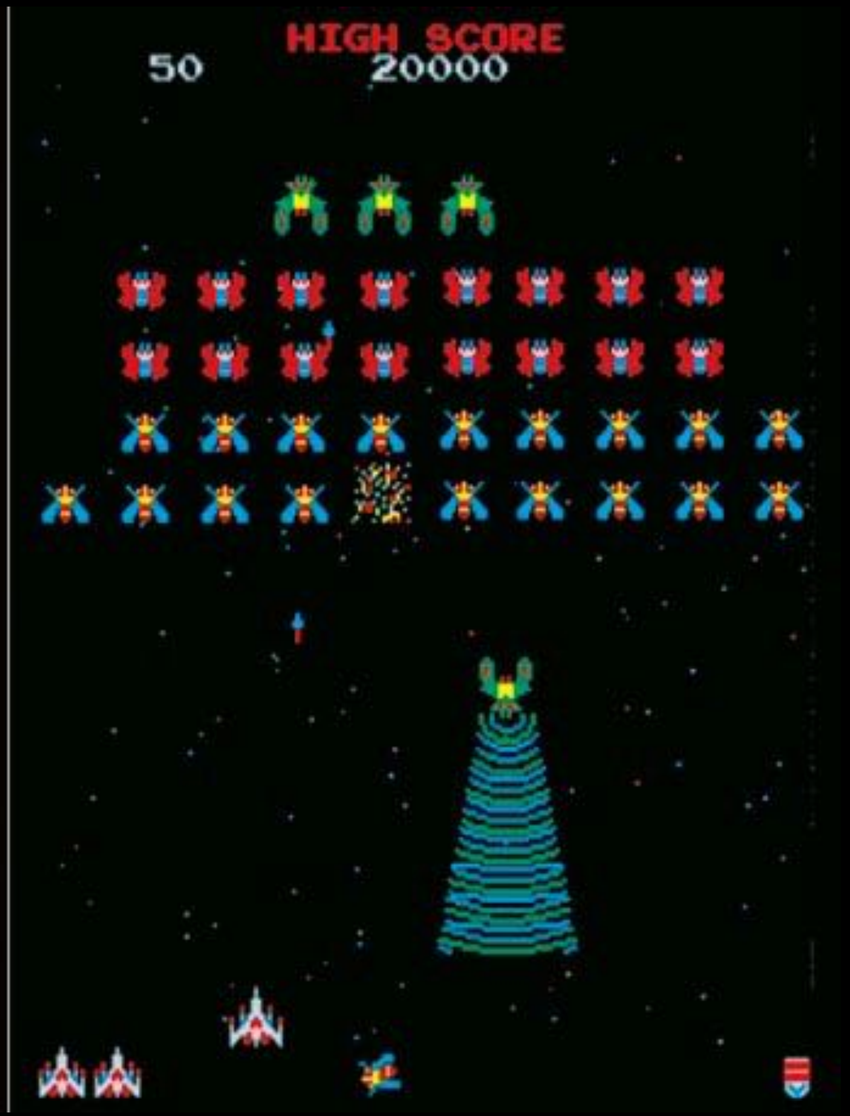
2560 * 1600 = 4,096,000 pixels

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.

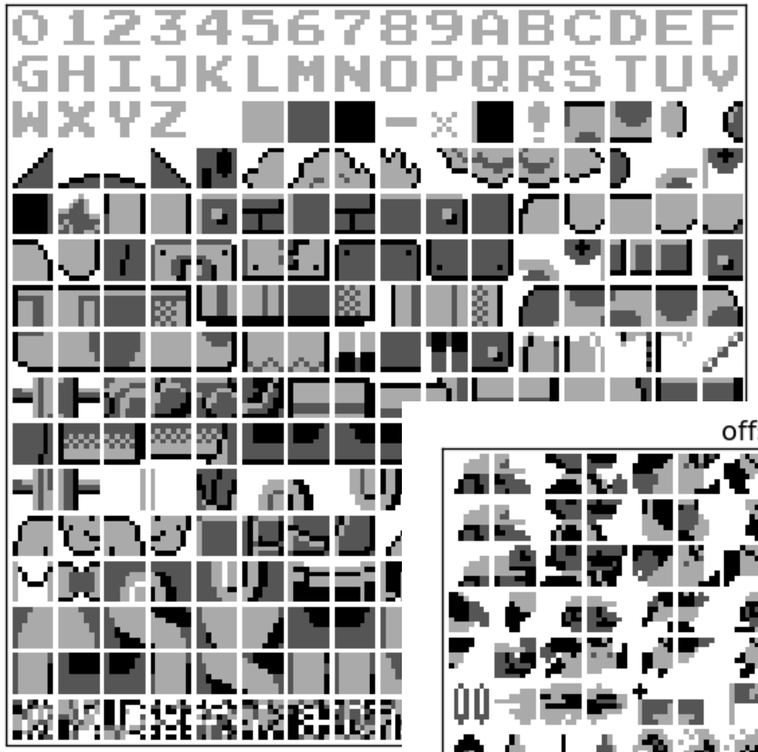
©1985 NINTENDO

1 PLAYER GAME

2 PLAYER GAME

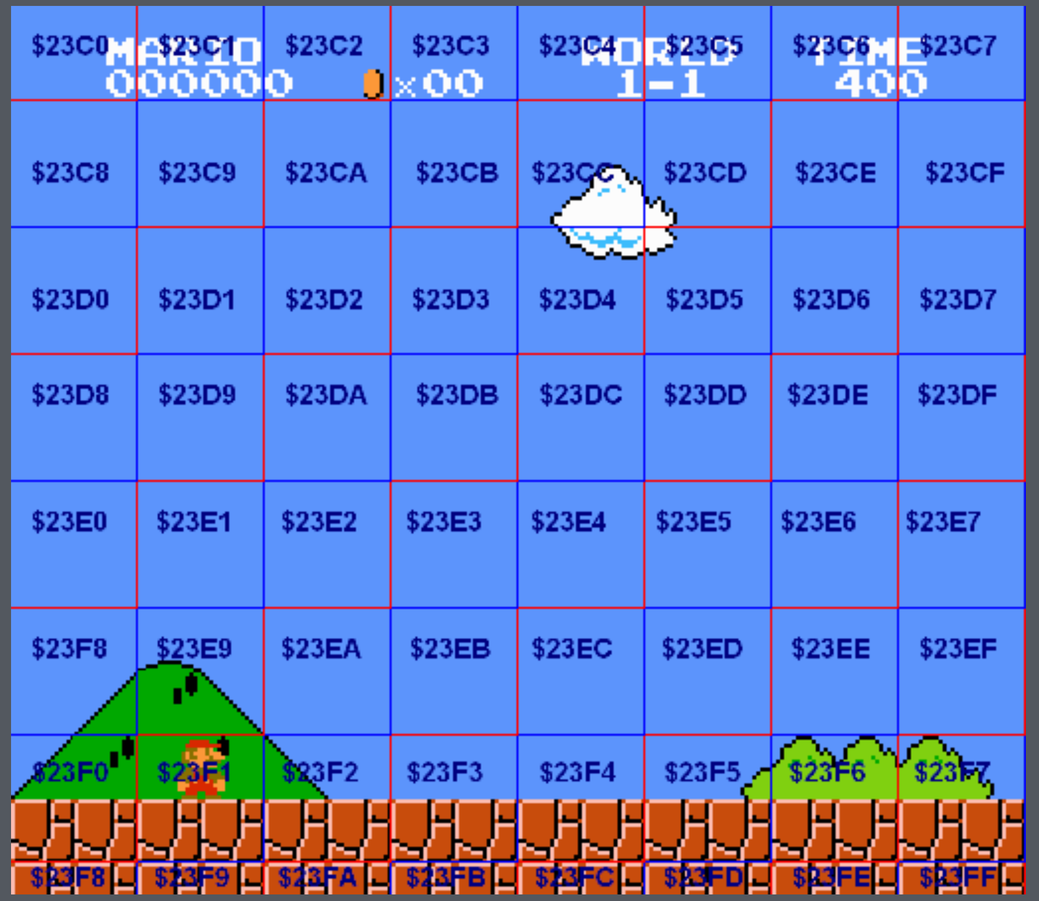
TOP- 000000

offset = 2305

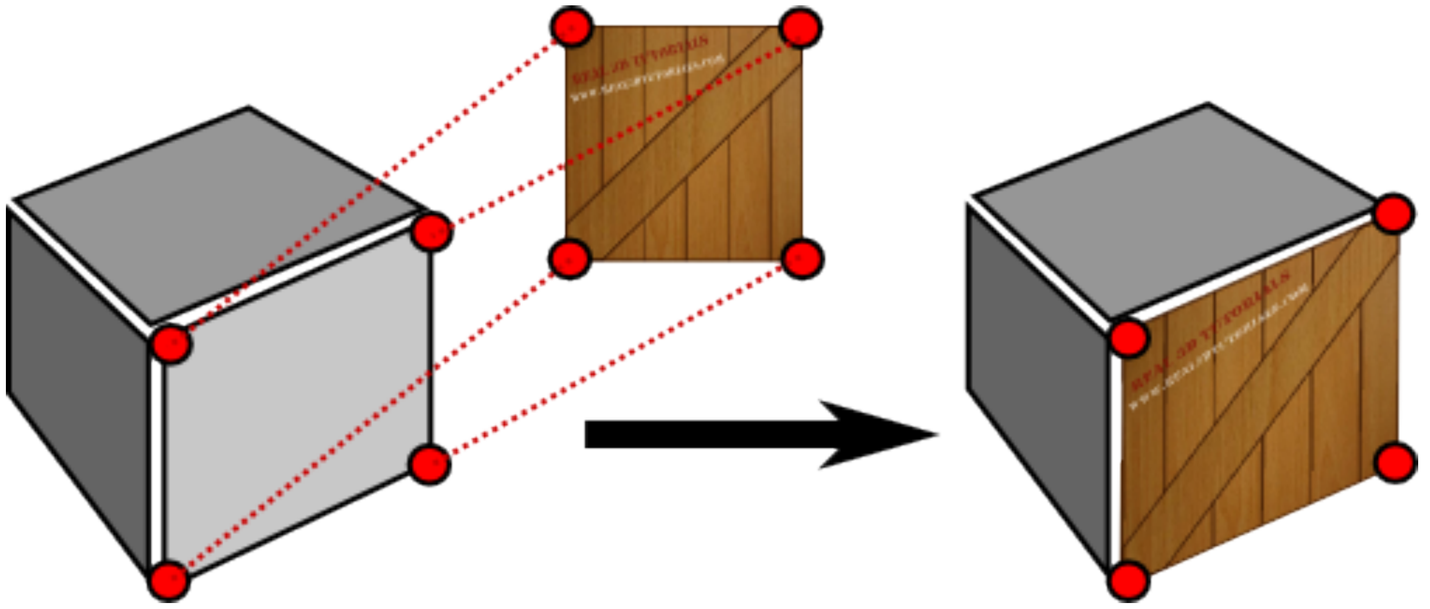


offset = 2049

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

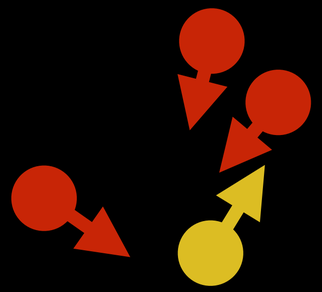

20
0
0
11
108

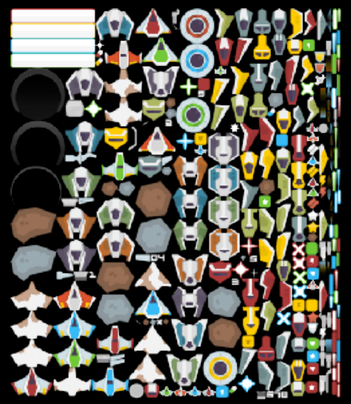

PARK
ONE W

Hardware Rendering

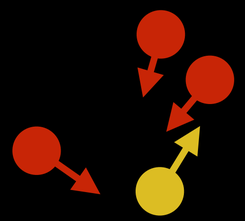
Hardware Rendering

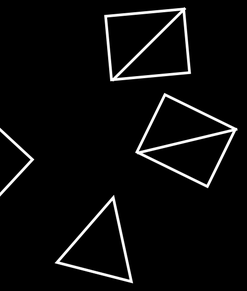
RAM

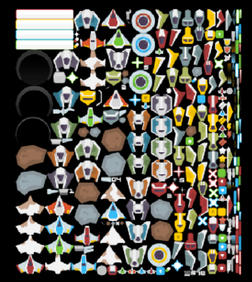




Video card



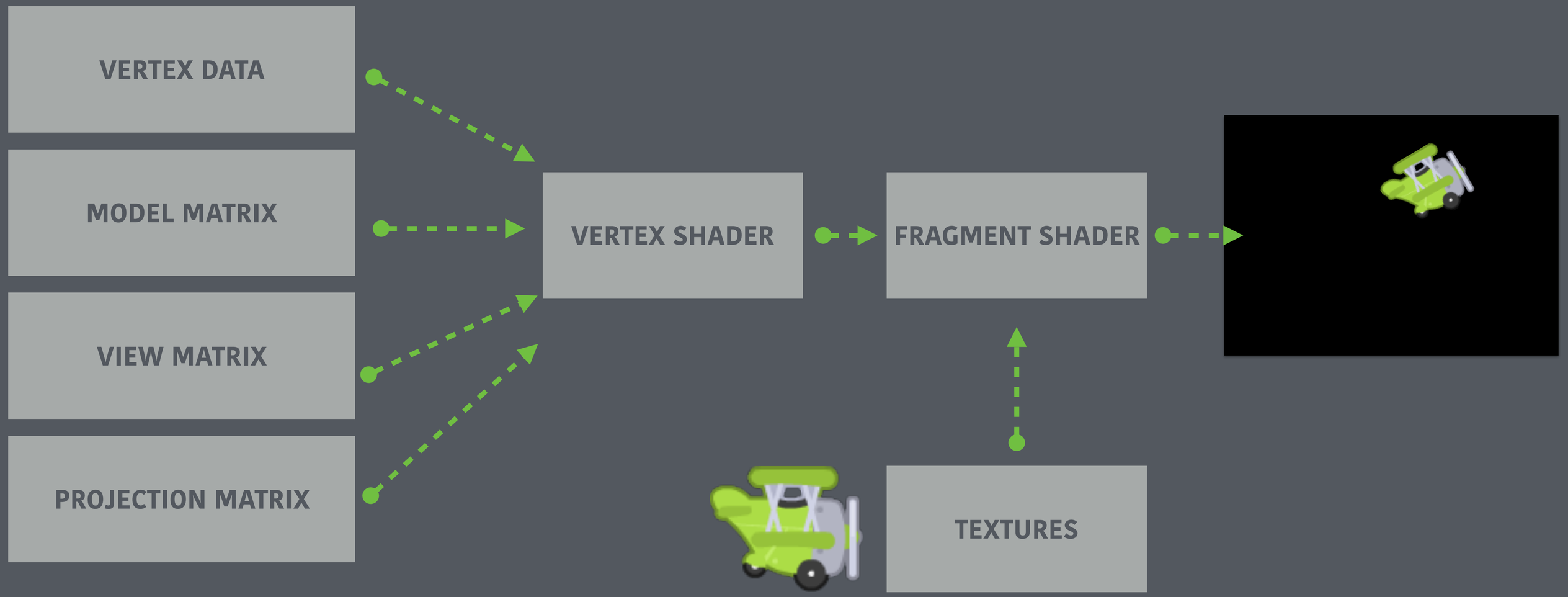




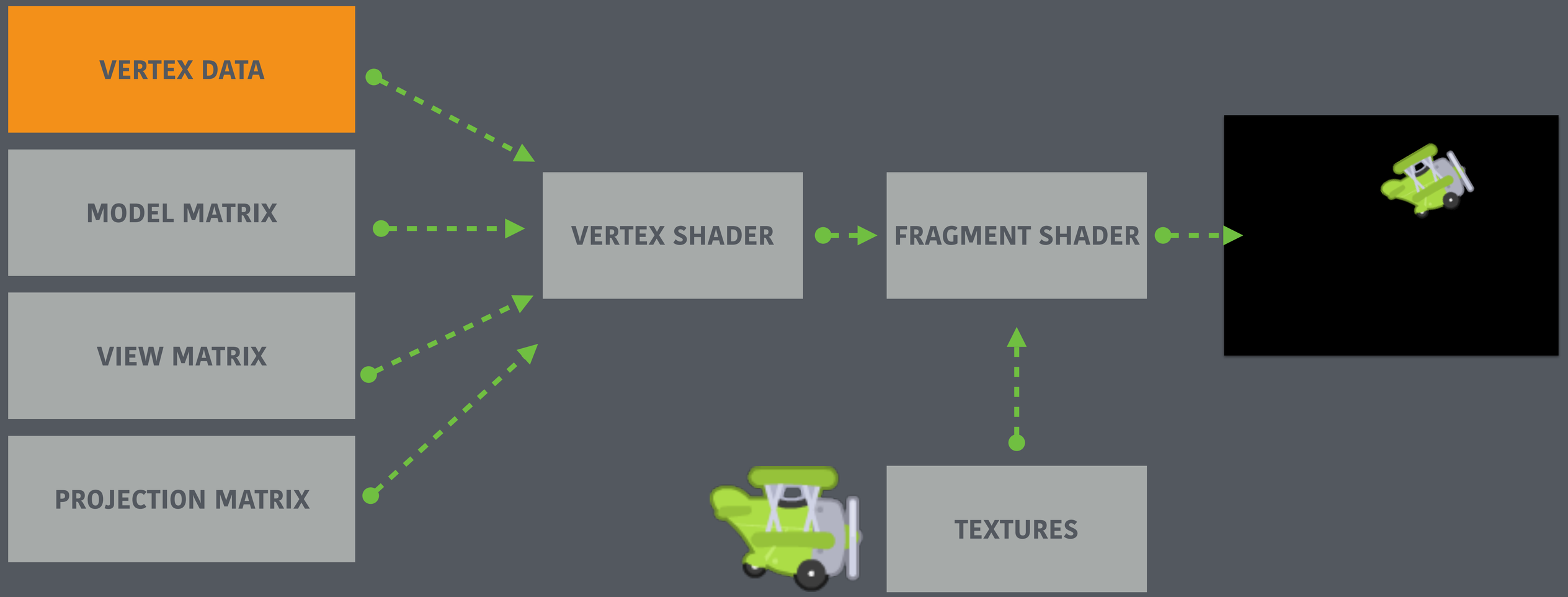
The GPU pipeline

What?
Where?
How?

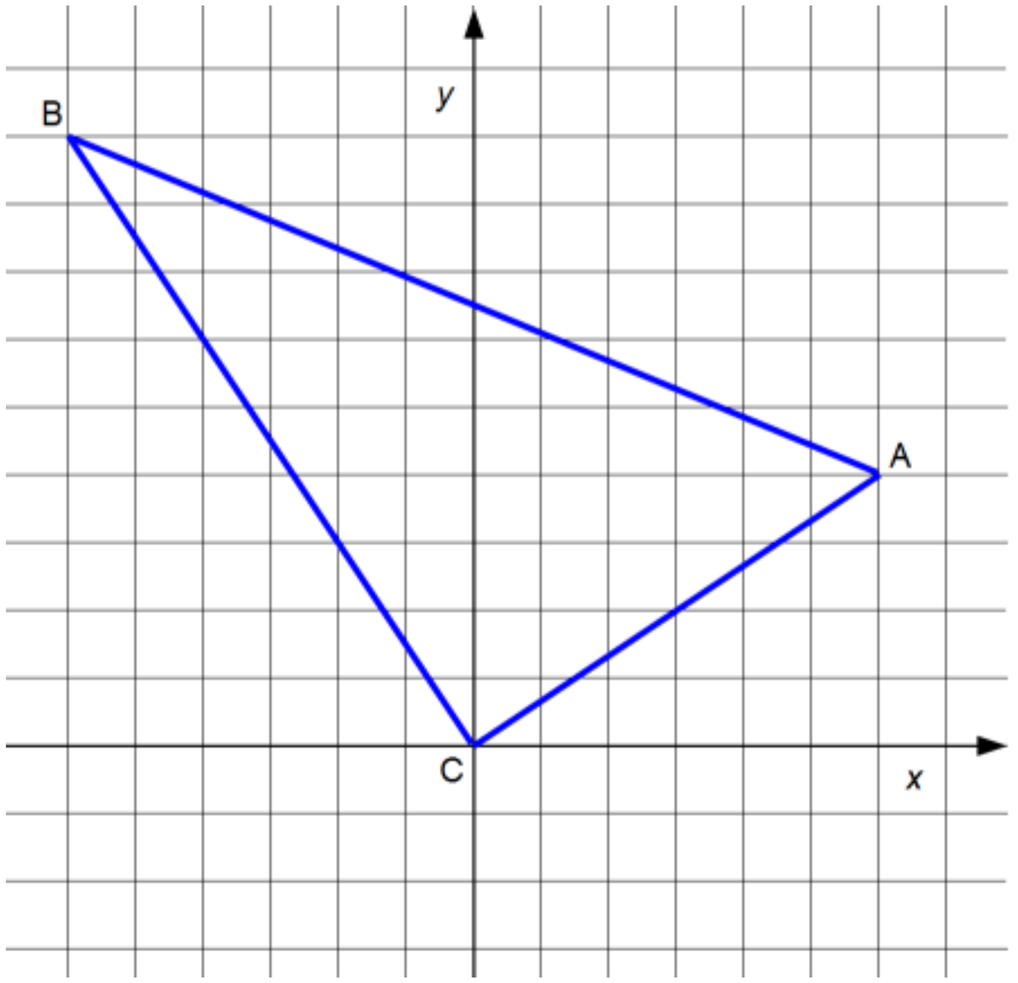
The **GPU** pipeline



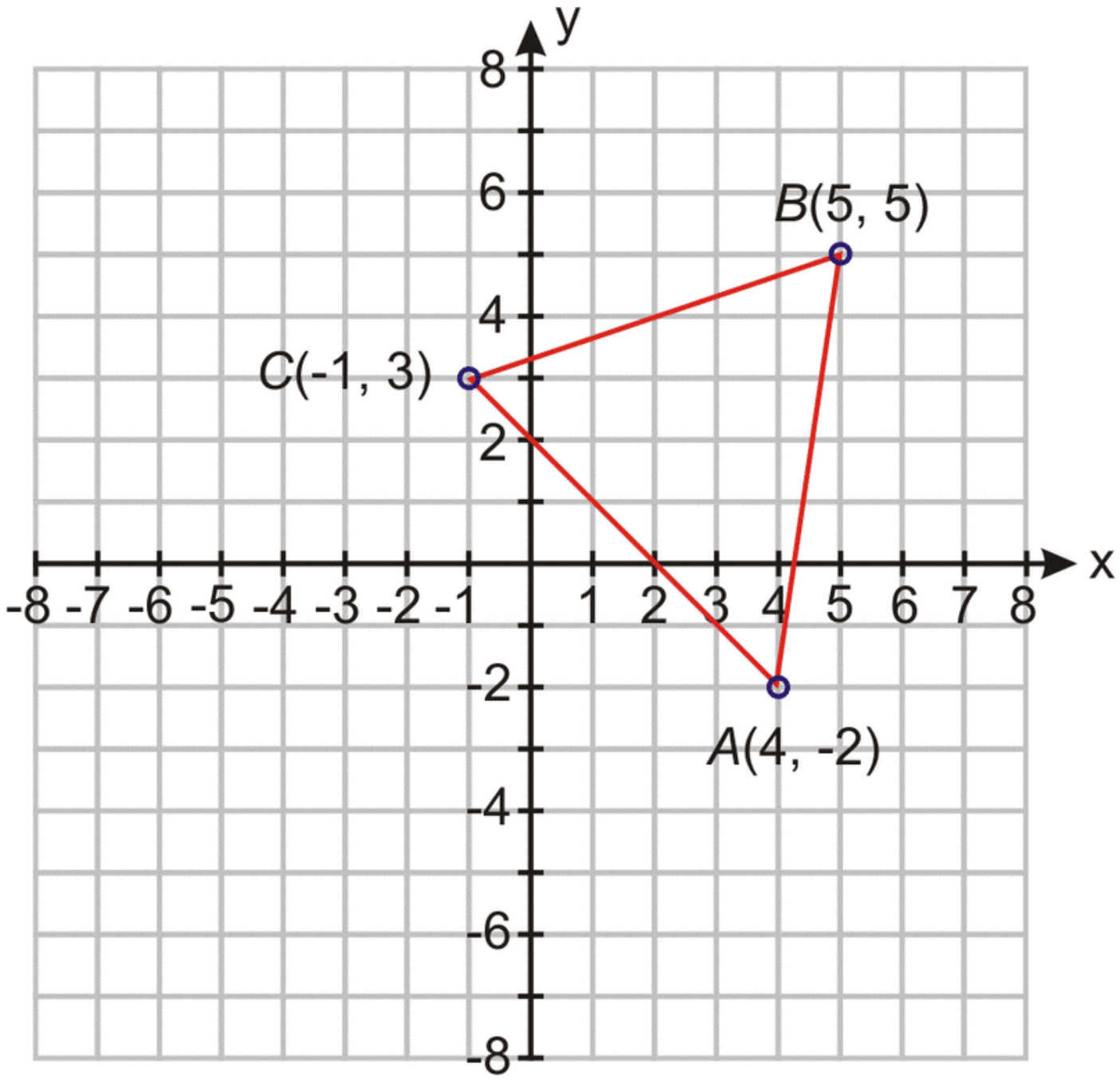
The **GPU** pipeline

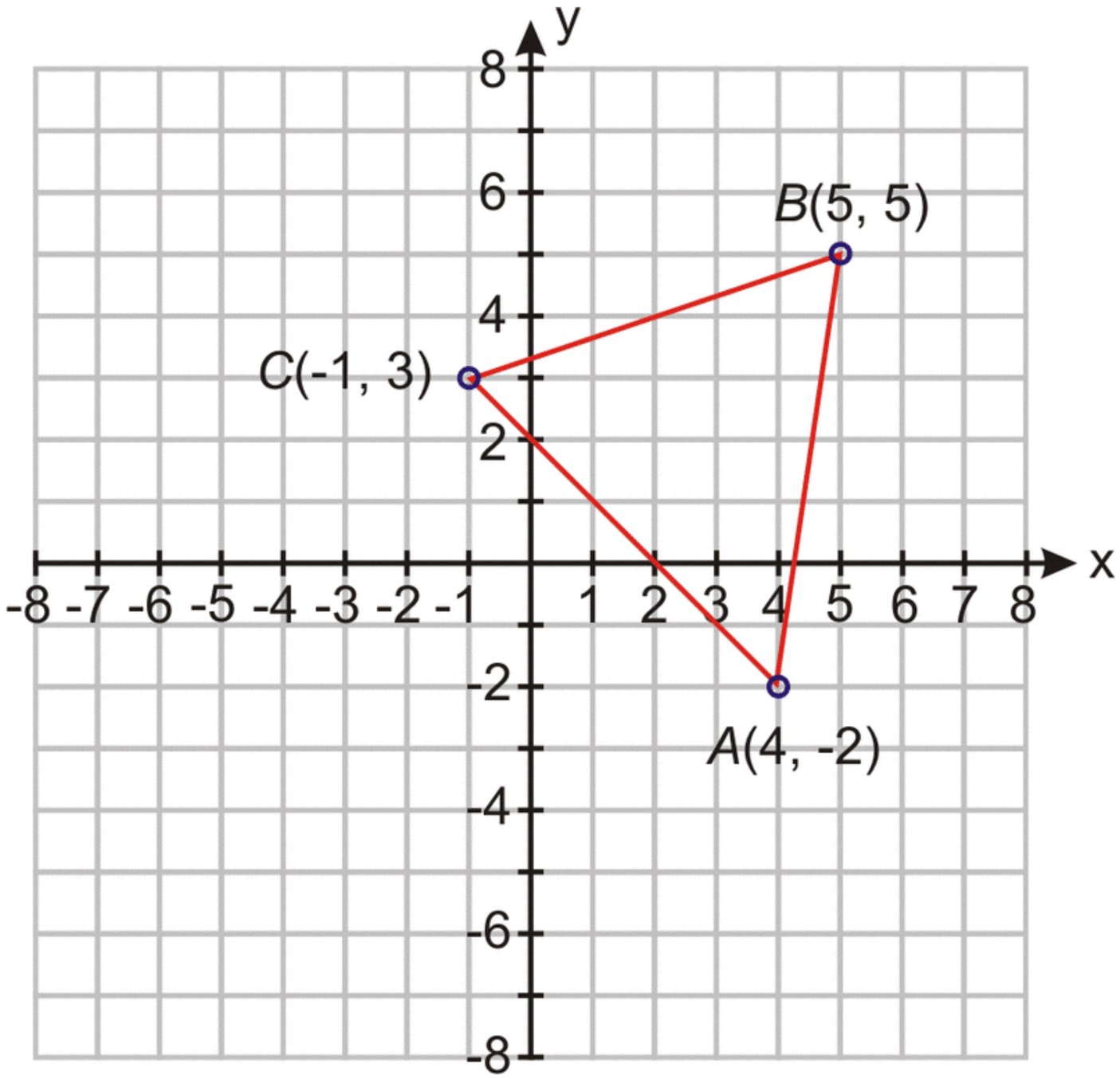


Vertex data


B
y
A
C
x

A polygon is defined by
points in space called vertices


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)

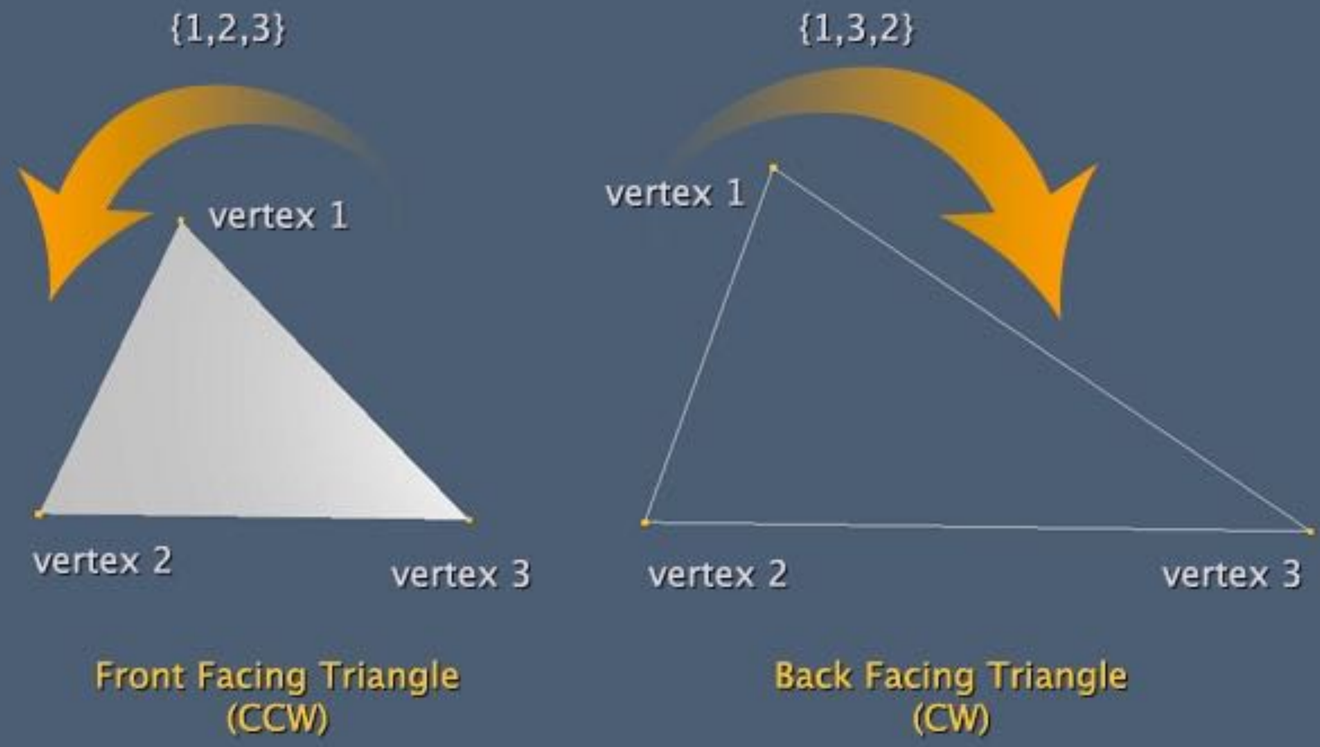


A: (4,−2)

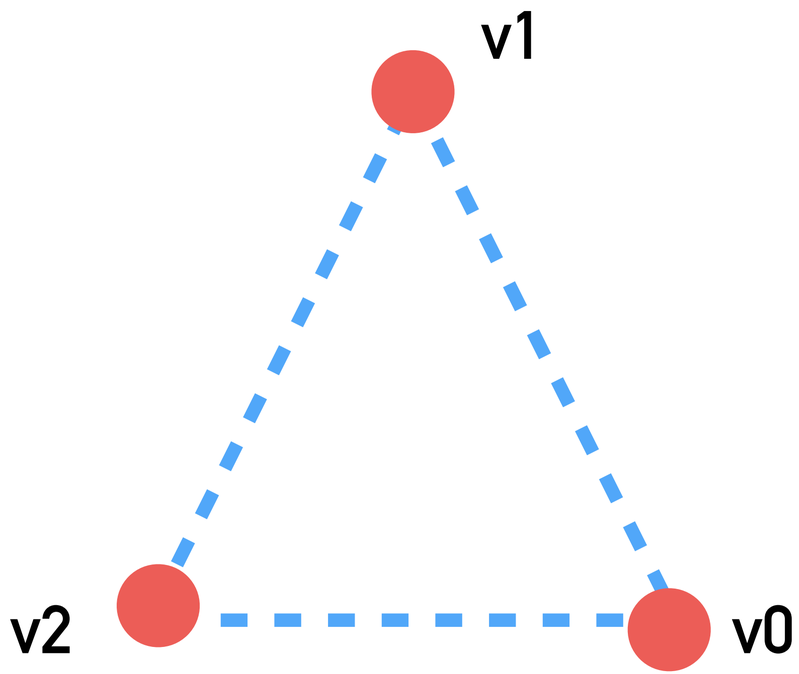
B: (5,5)

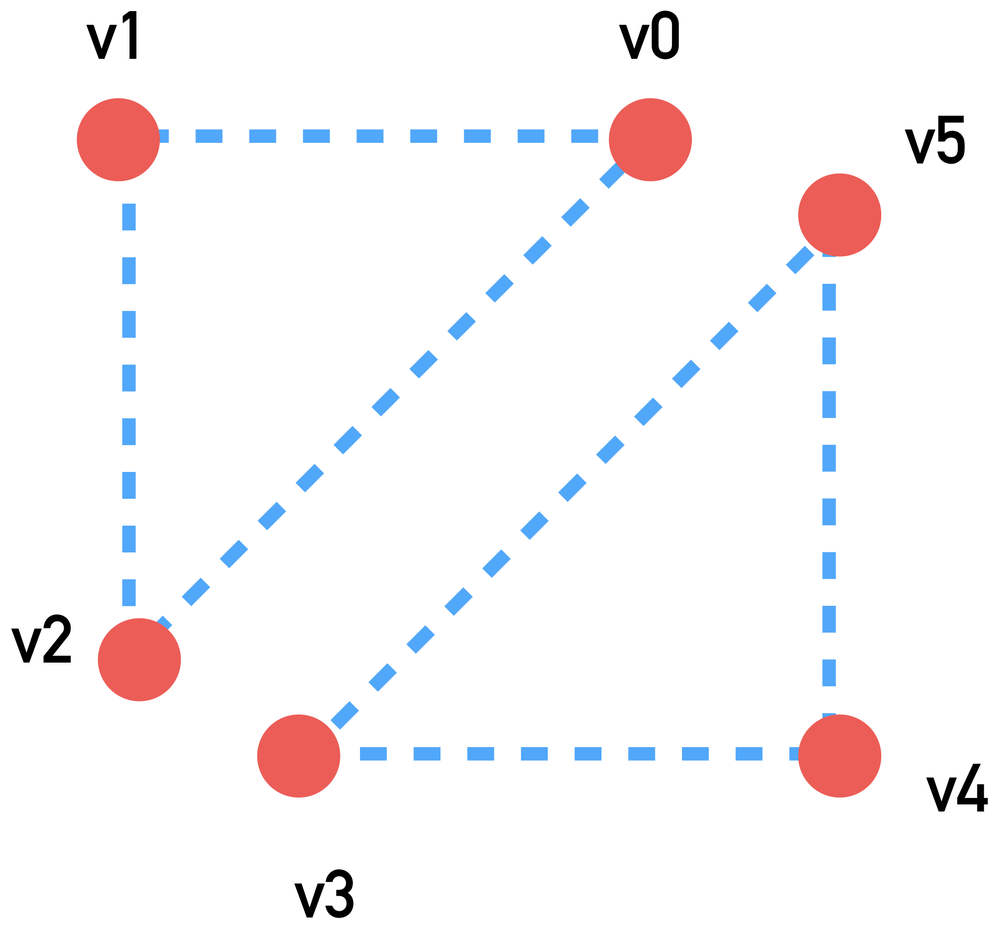
C: (−1, 3)

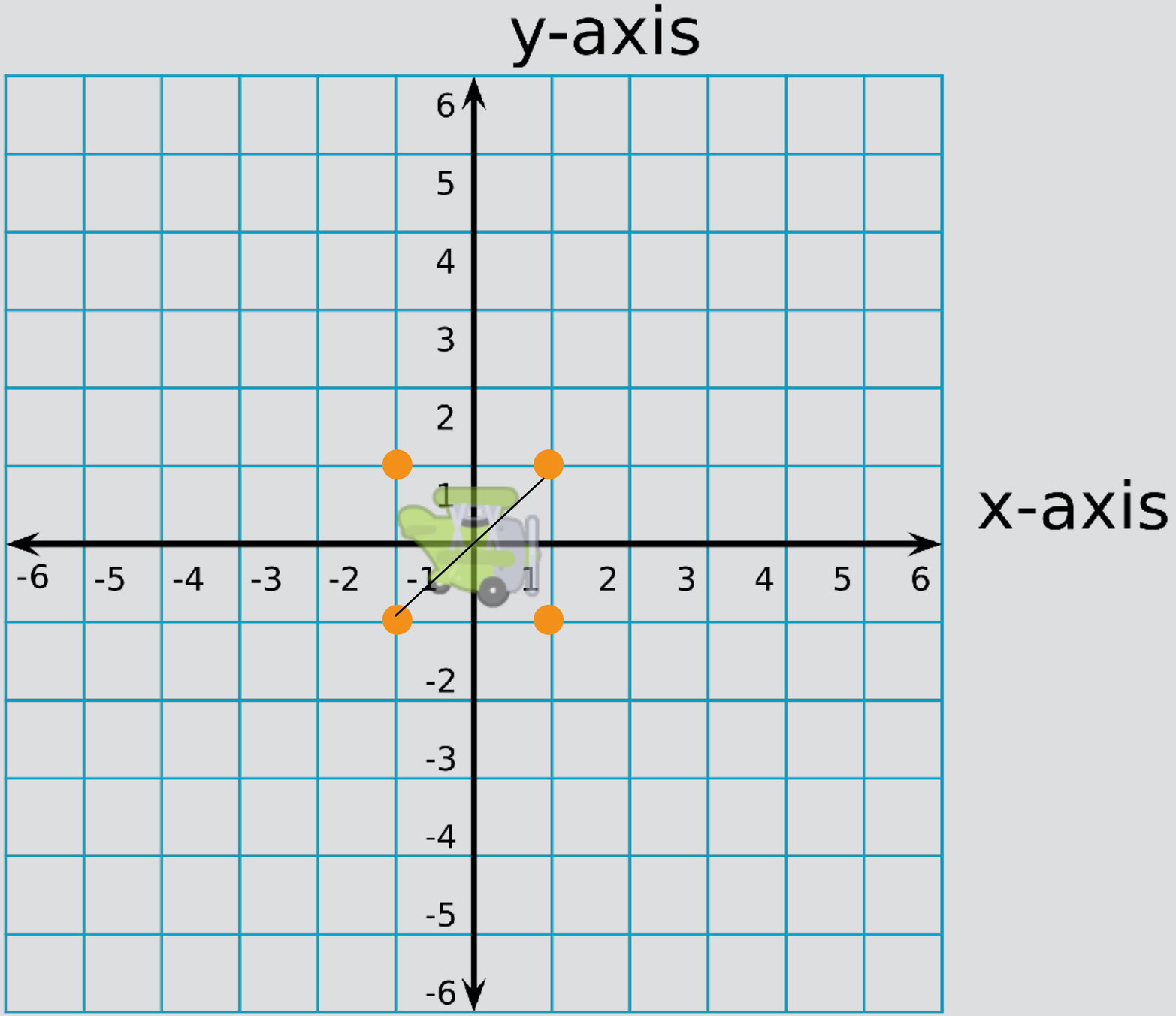
Polygons are **one-sided** and the side is defined by the **order of the vertices**



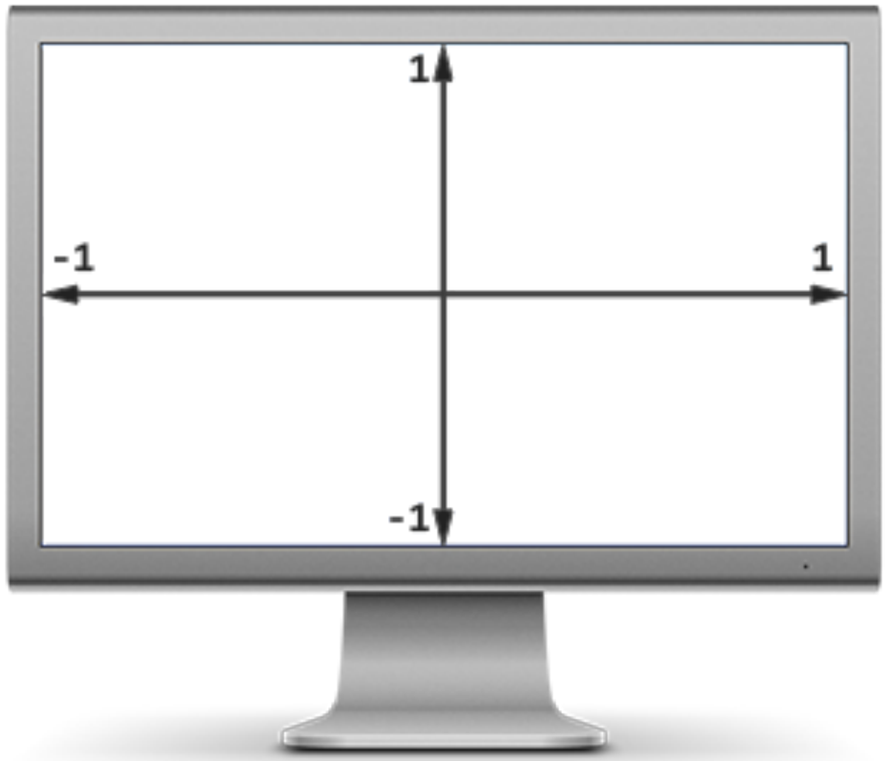
Counter-clockwise order!


v1
v2
v0

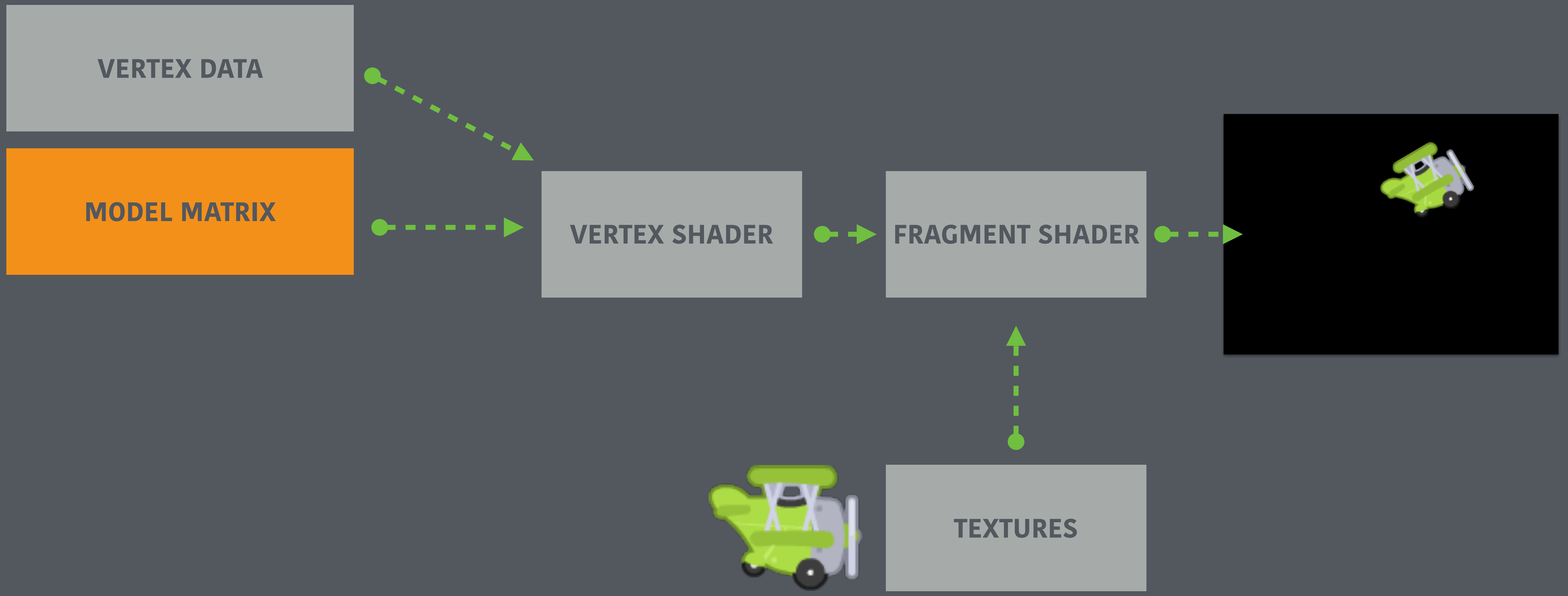

v1
v0
v5
v2
v4
v3


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

Normalized Device Coordinates



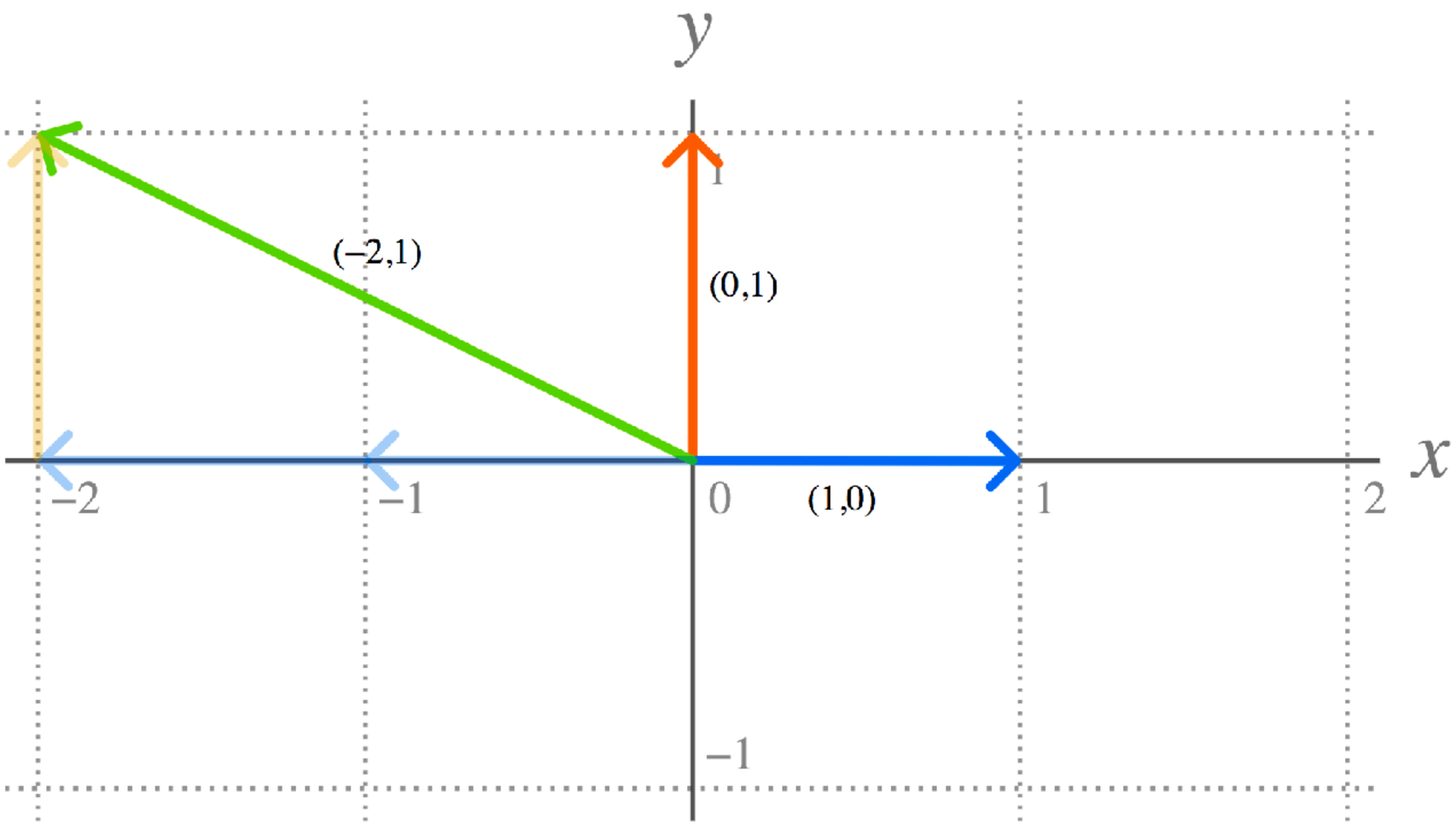
The **GPU** pipeline

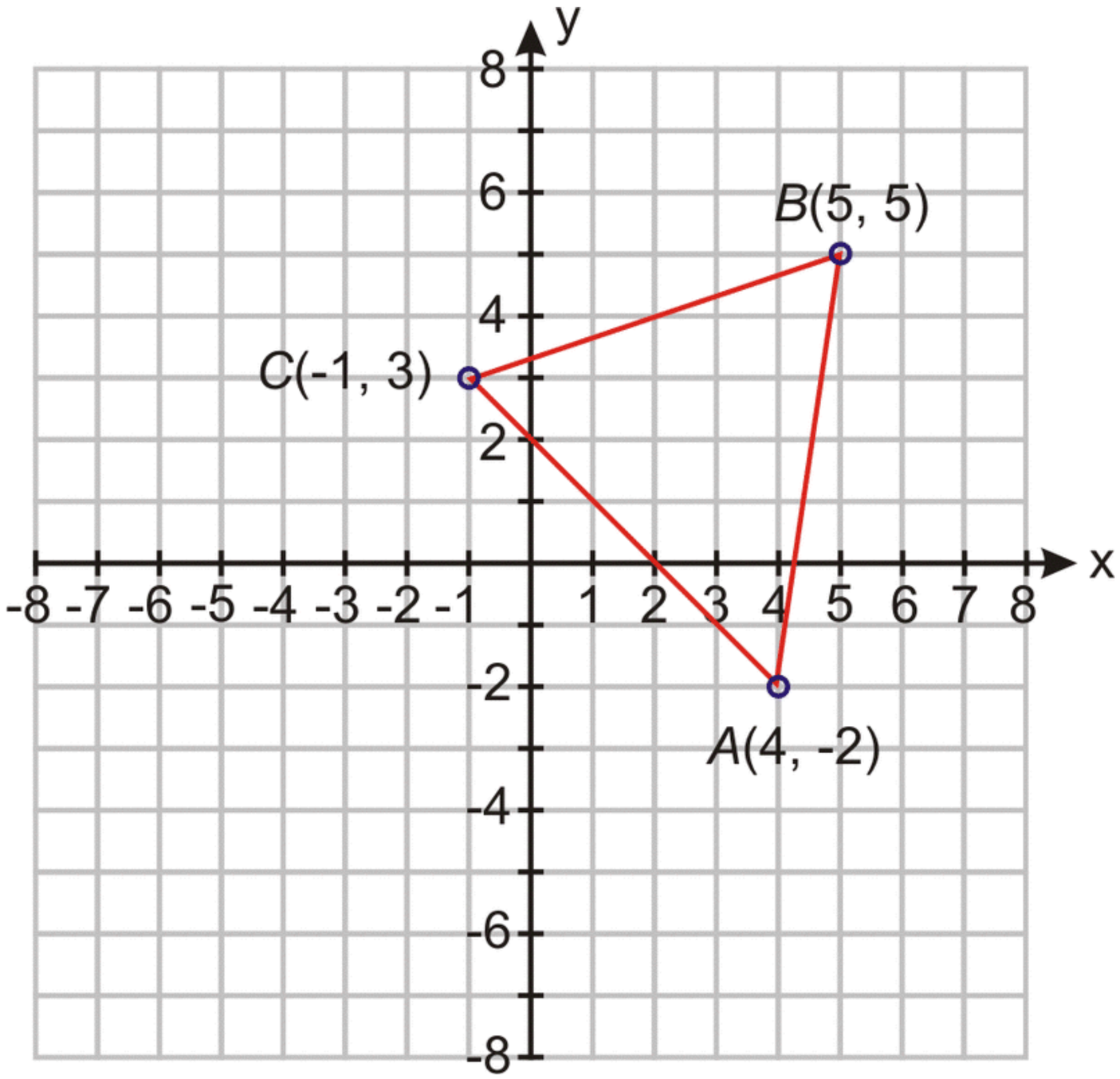


The model matrix

Vectors.

Basis vectors.



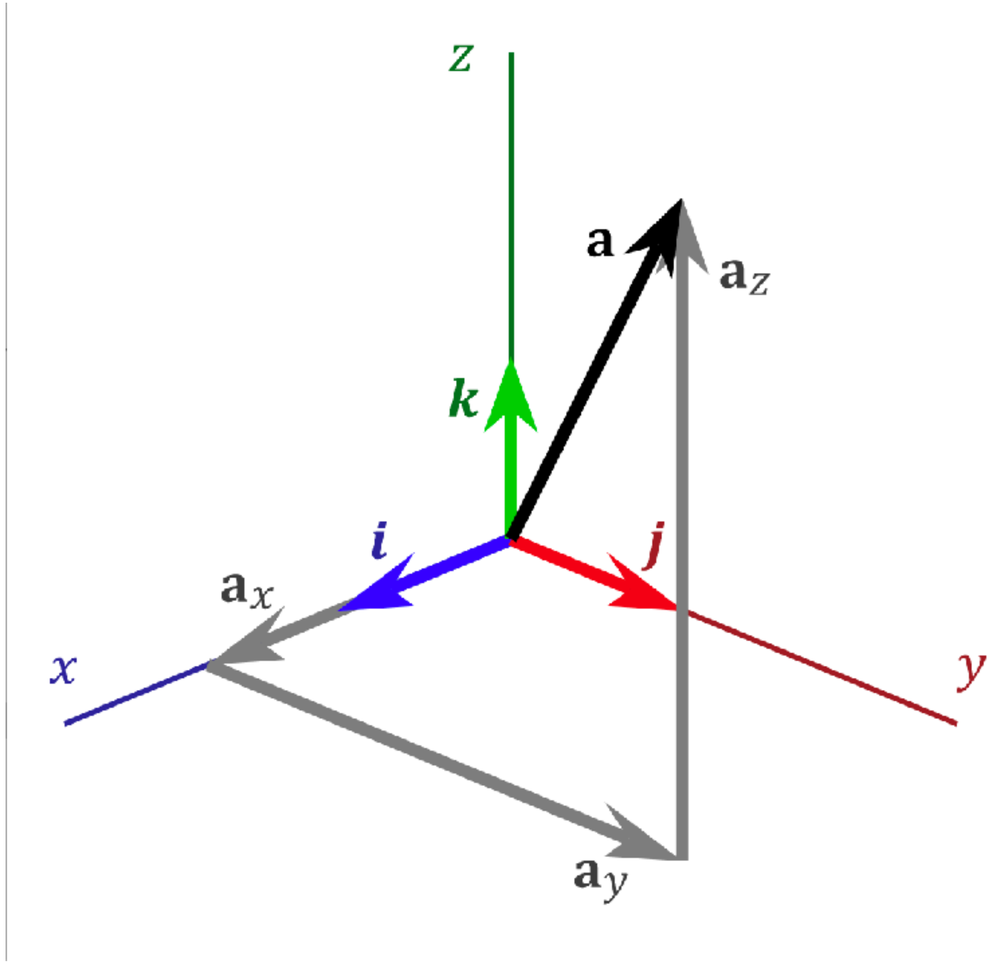


A: (4,−2)

B: (5,5)

C: (−1, 3)

Basis vectors.



$$\vec{v} = \sum_i c_i \vec{b}_i .$$

$$\vec{v} = \sum_i c_i \vec{b}_i = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} .$$

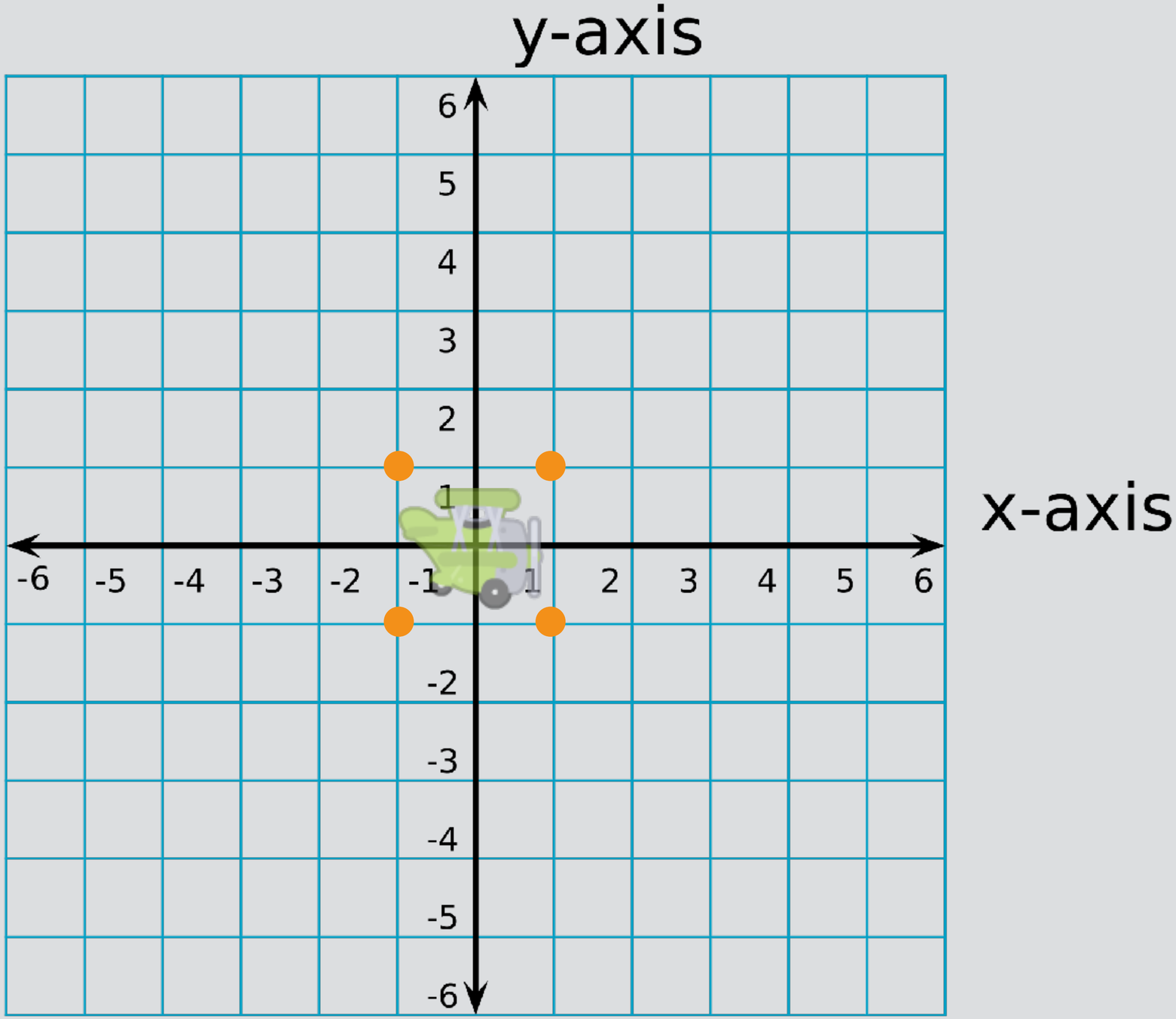
Transformation matrix

A linear transformation matrix represents a linear transformation in space (rotate and scale).

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

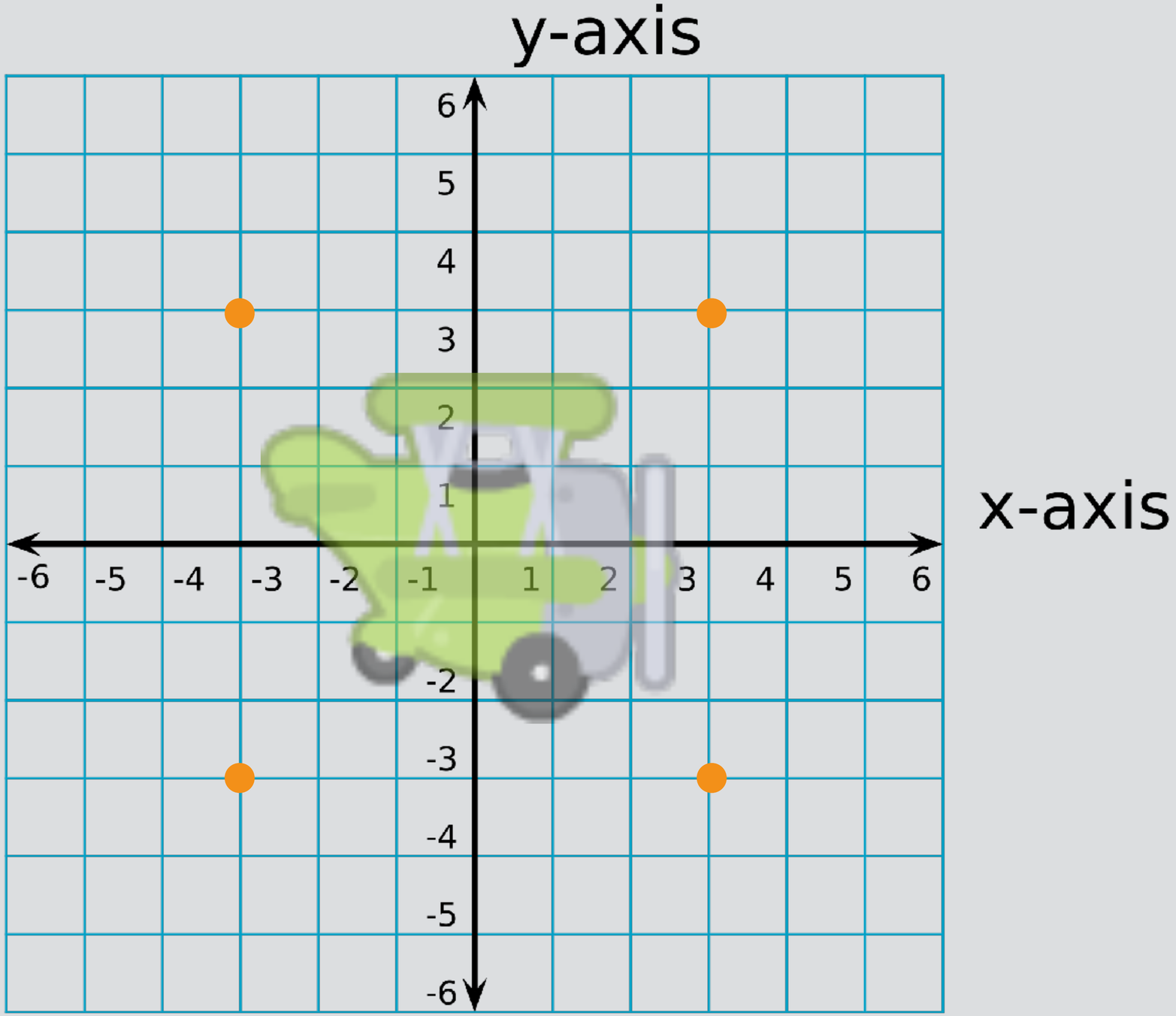
Identity

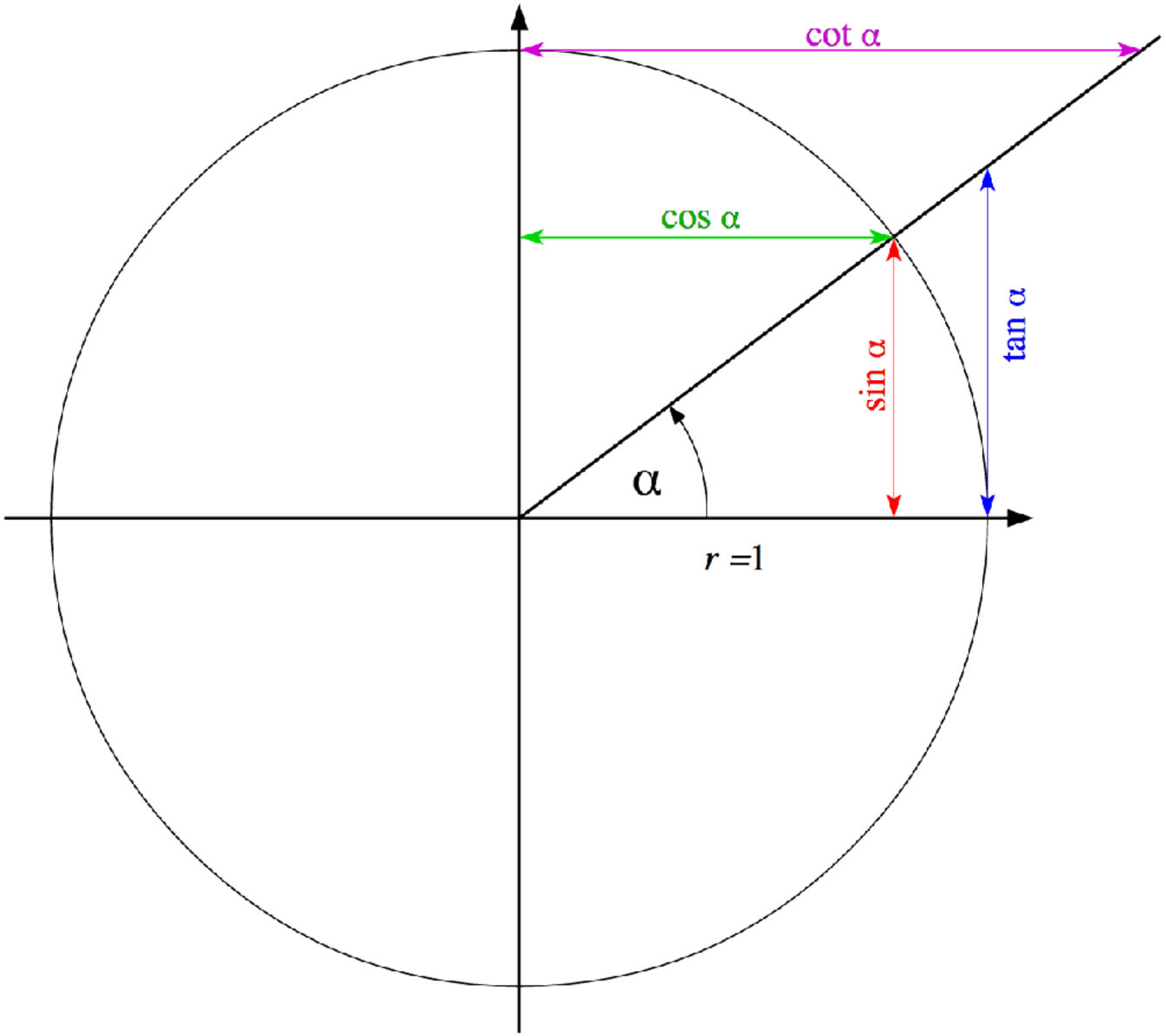
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Scale

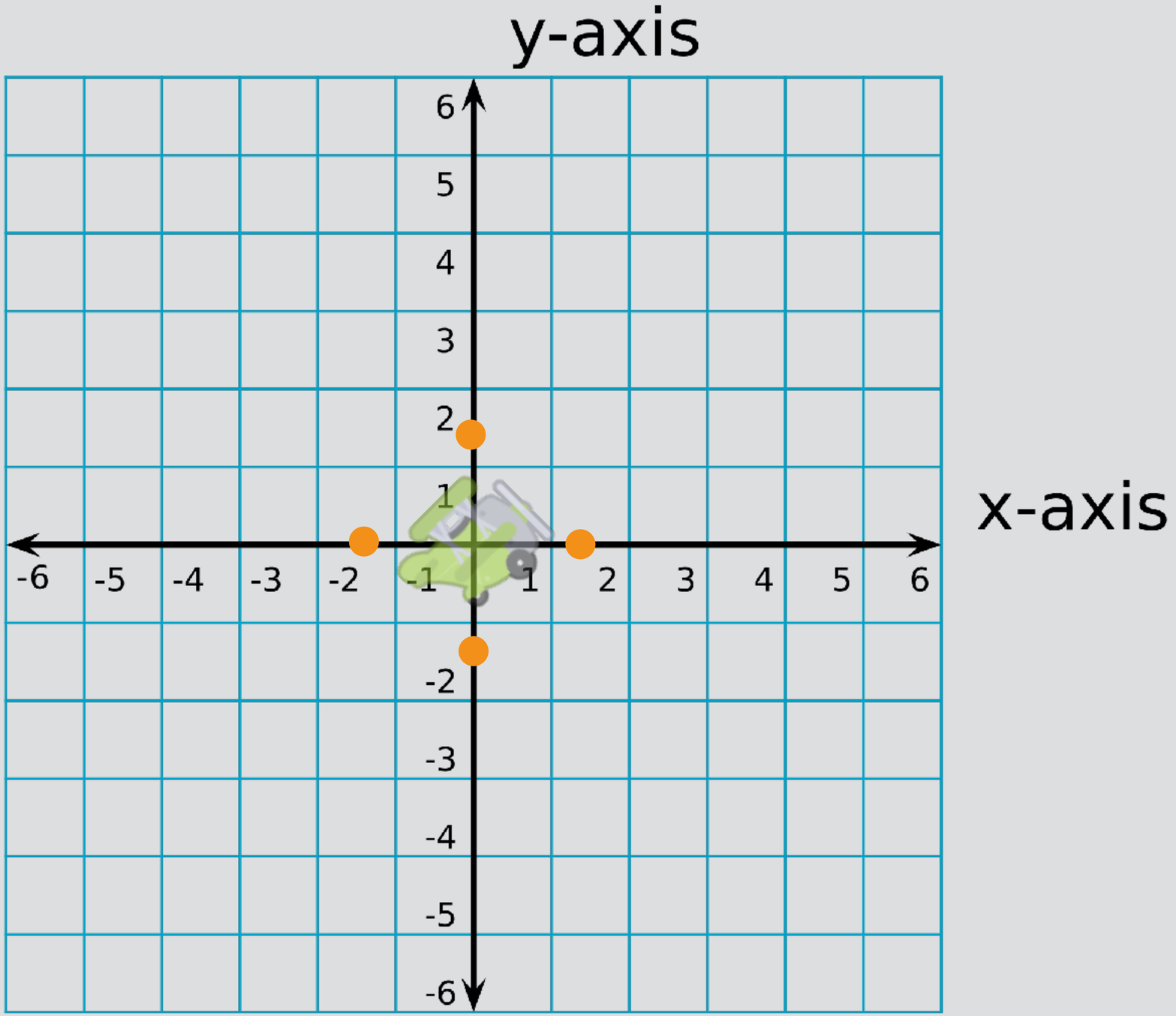
$$\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & Sz \end{bmatrix}$$

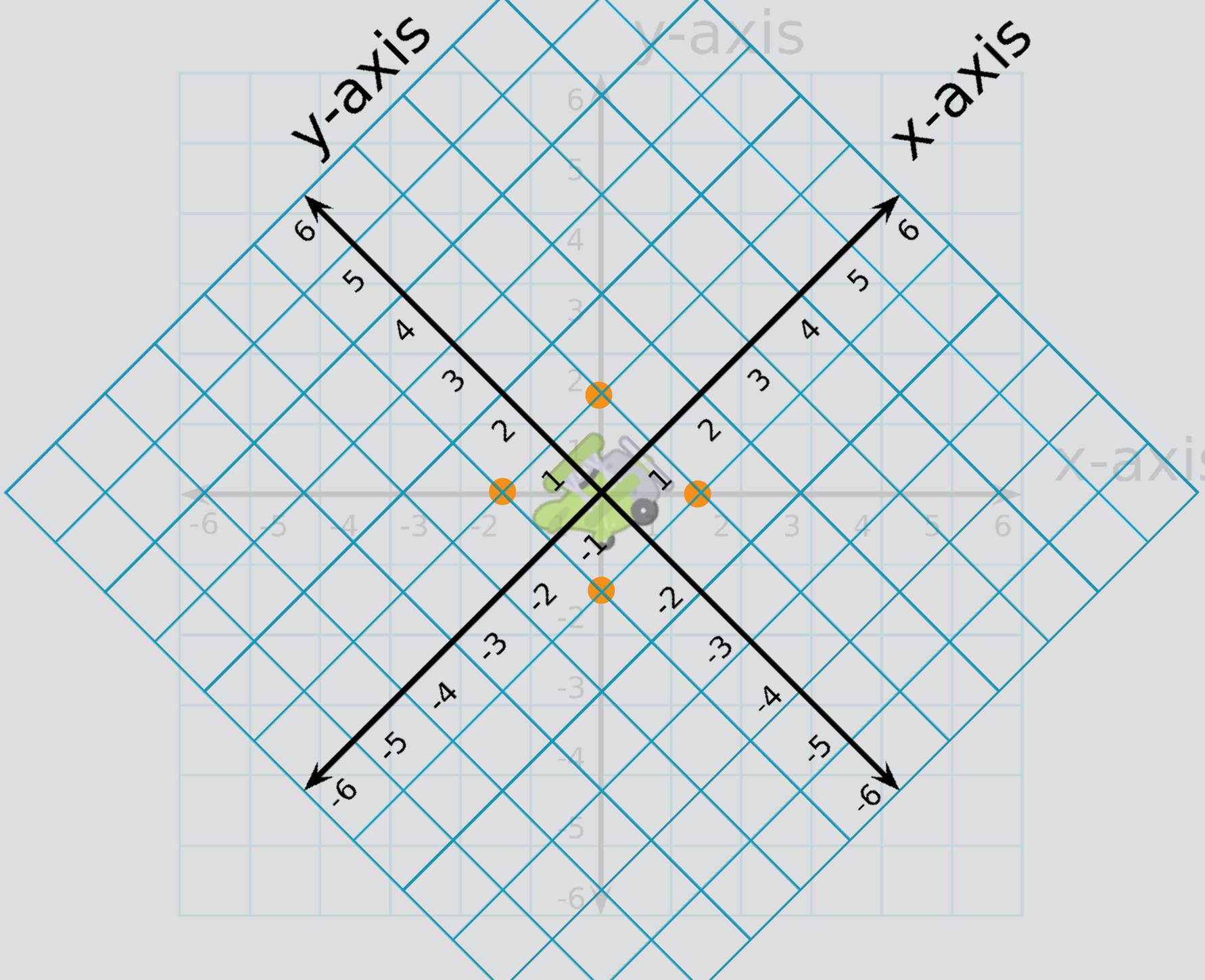



cot α
cos α
sin α
tan α
α
r =1

Rotate

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



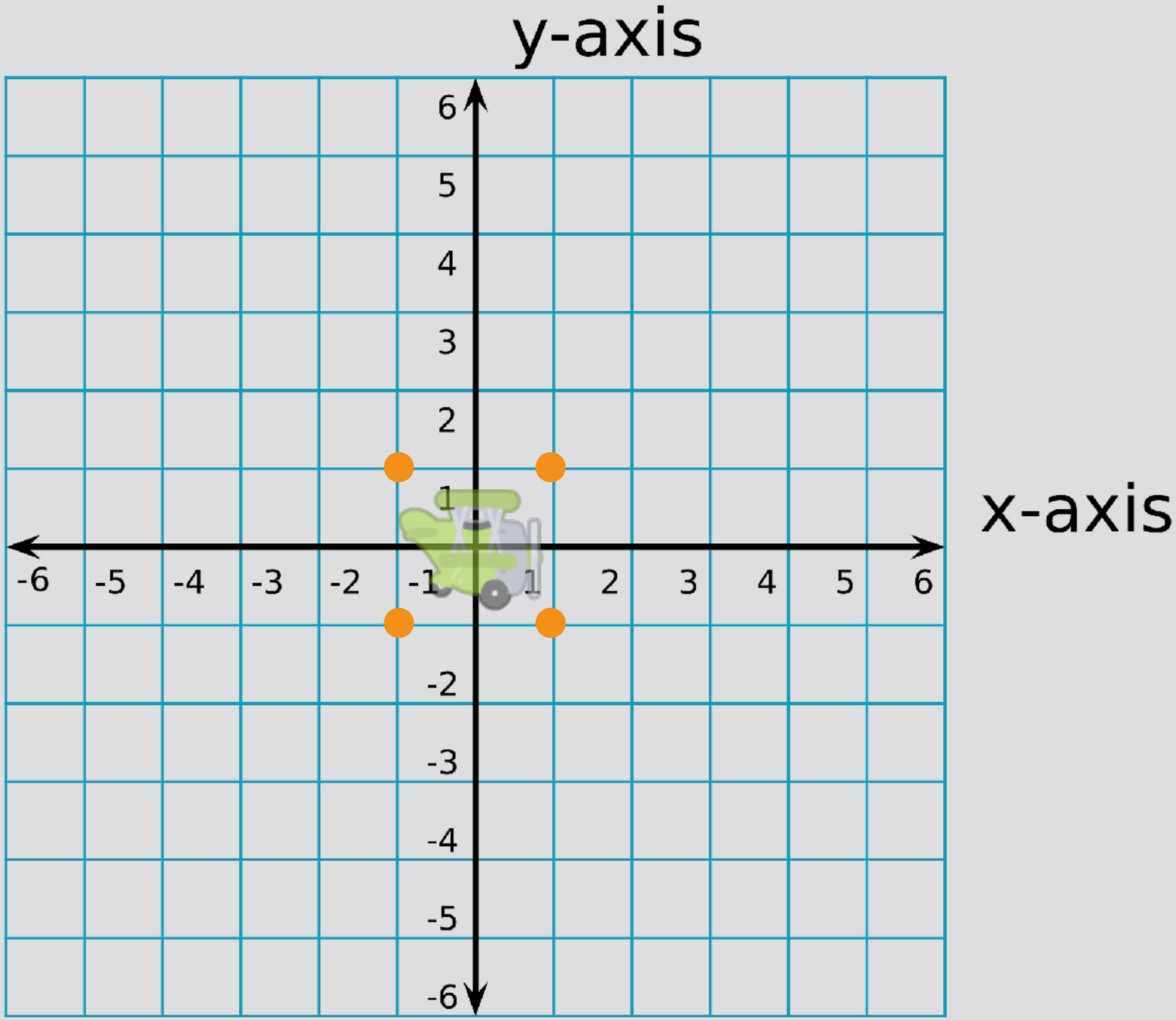

y-axis
x-axis
6
5
4
3
2
1
-1
-2
-3
-4
-5
-6

An affine transformation matrix combines a linear transformation matrix with a translation using homogeneous coordinates.

Affine Identity

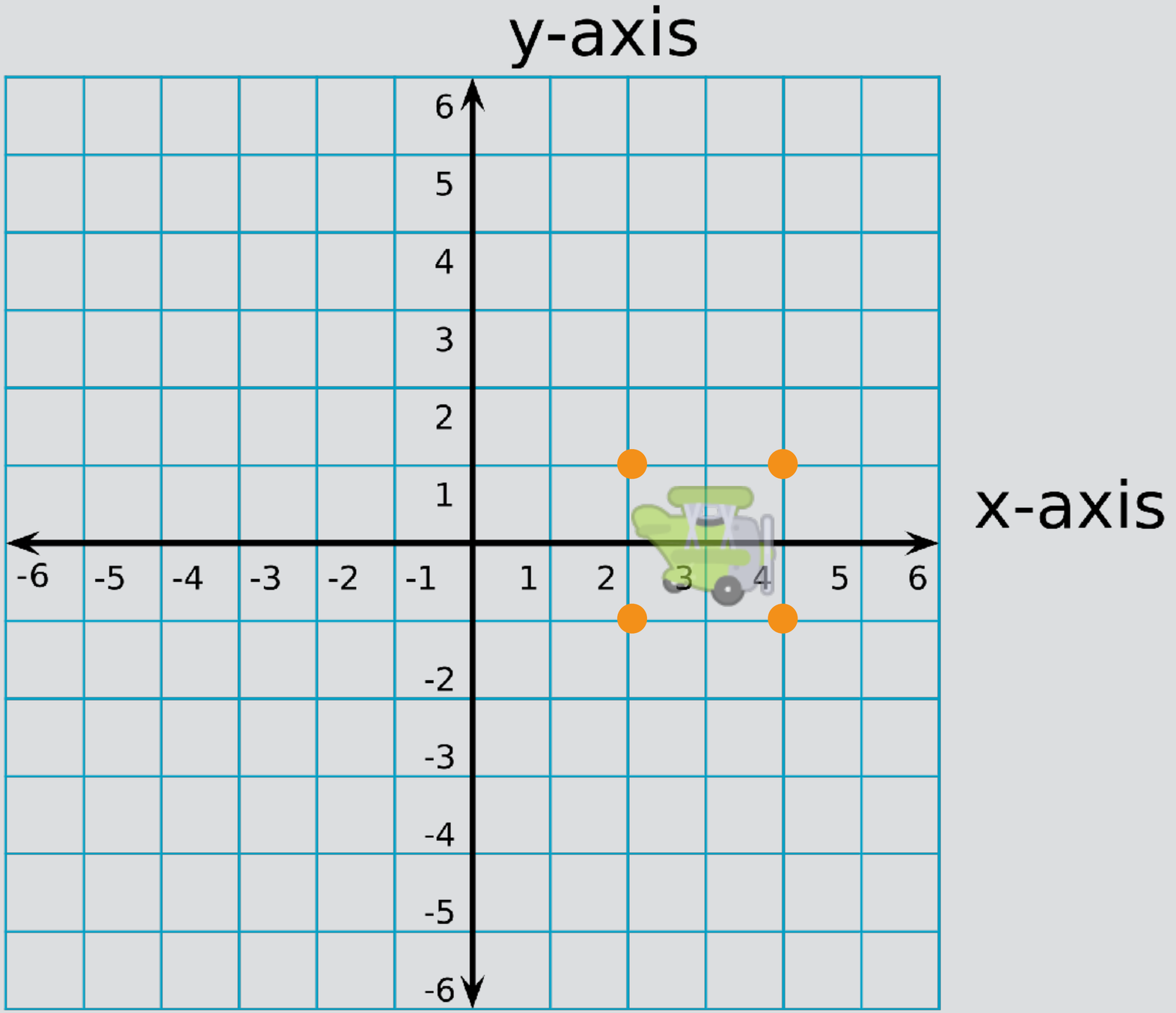
Linear part

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



Affine Translation

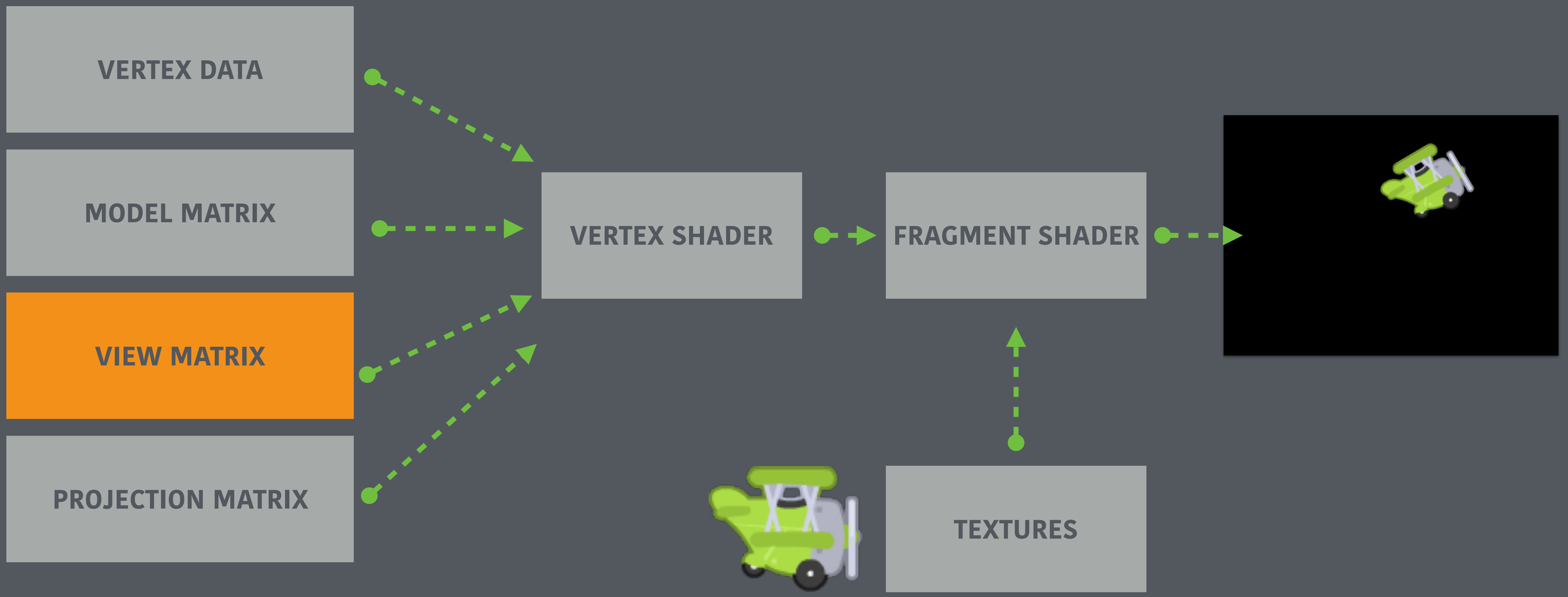
$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



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

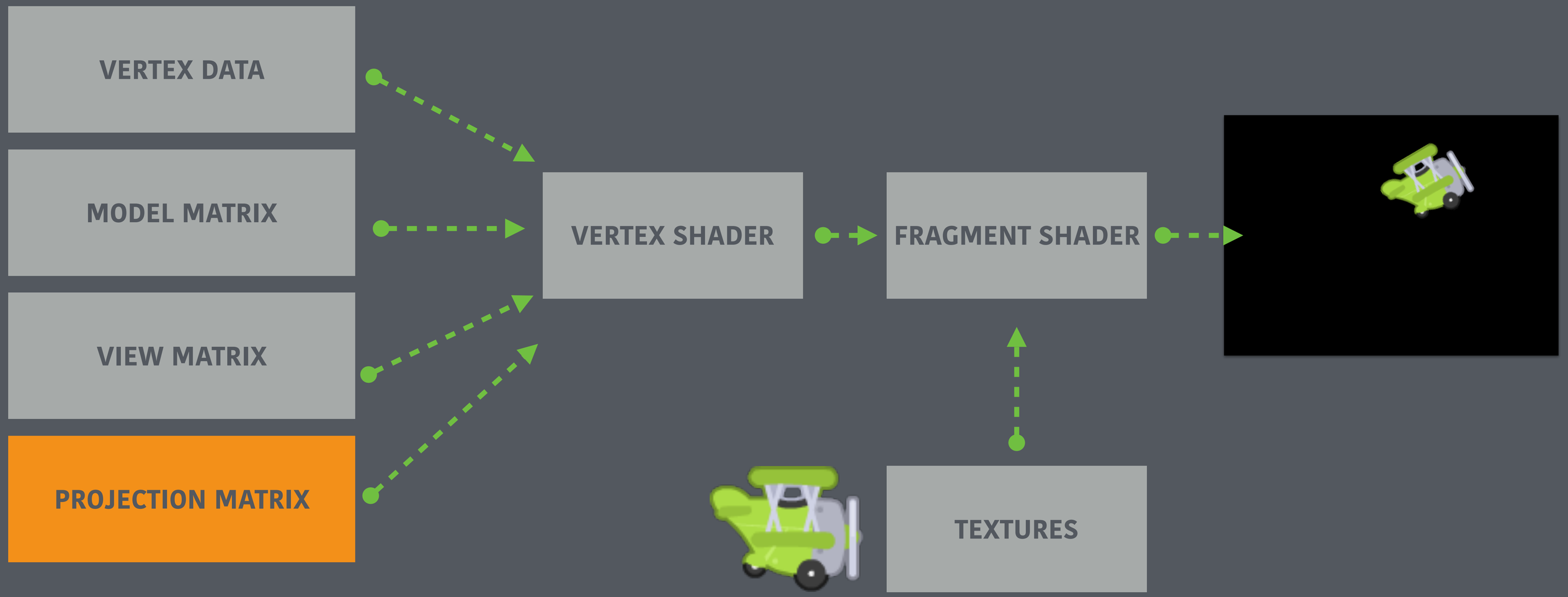


View matrix

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)

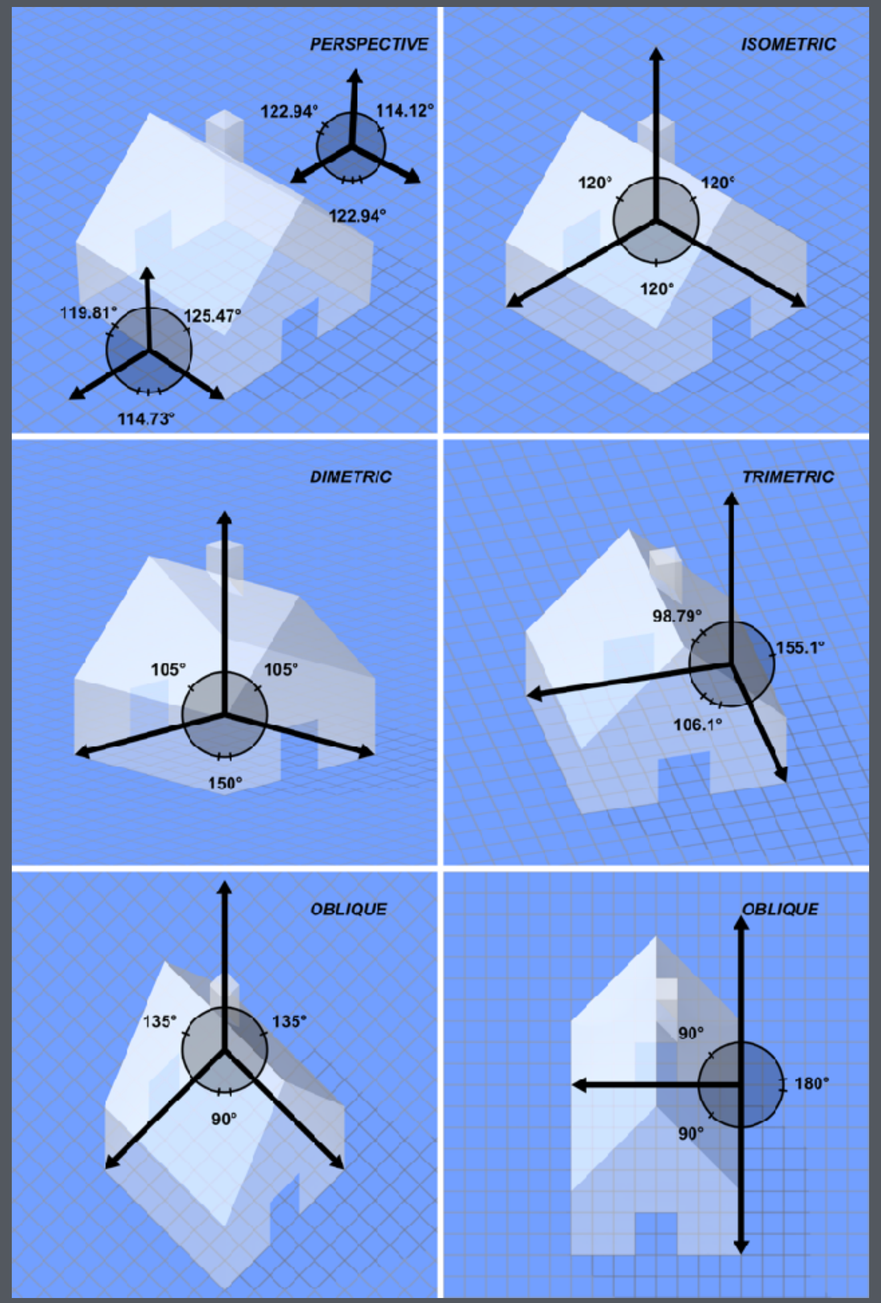
The **GPU** pipeline



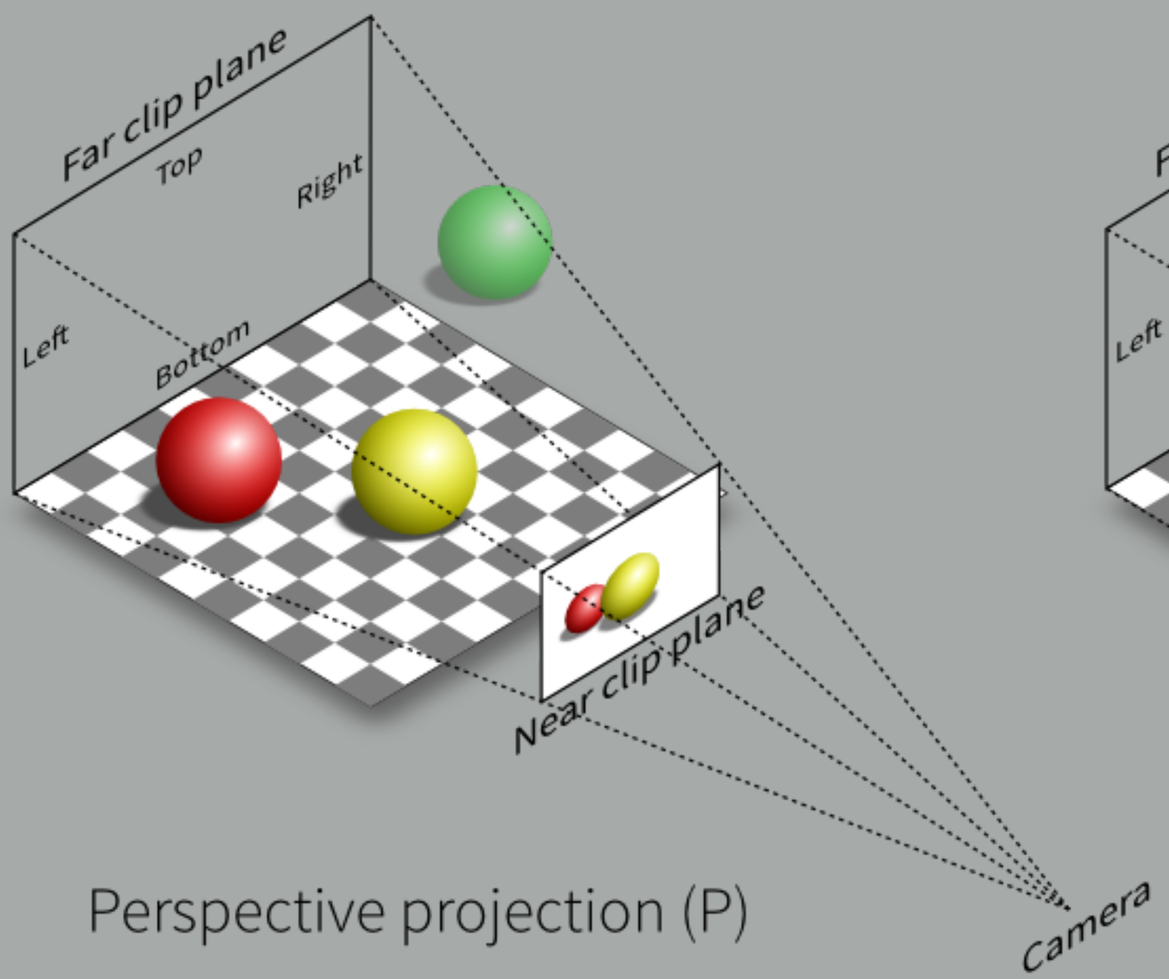
Projection matrix

What is projection?

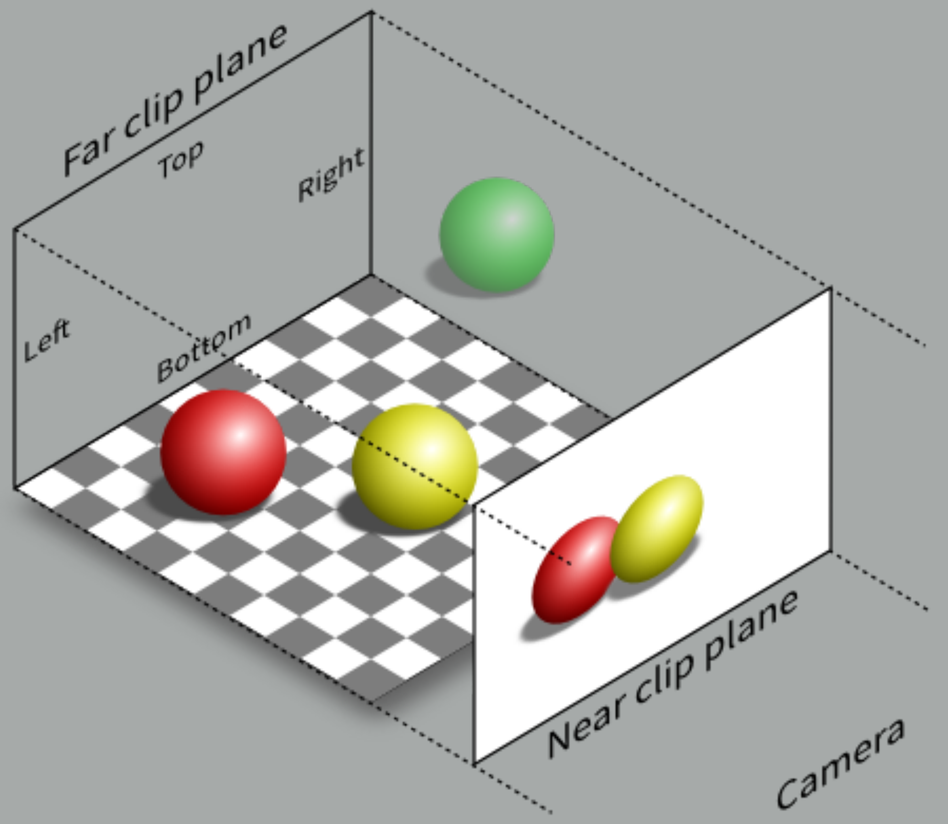
A means of representing a **three-dimensional** object in **two-dimensions**.



Perspective vs. **Orthographic** projection

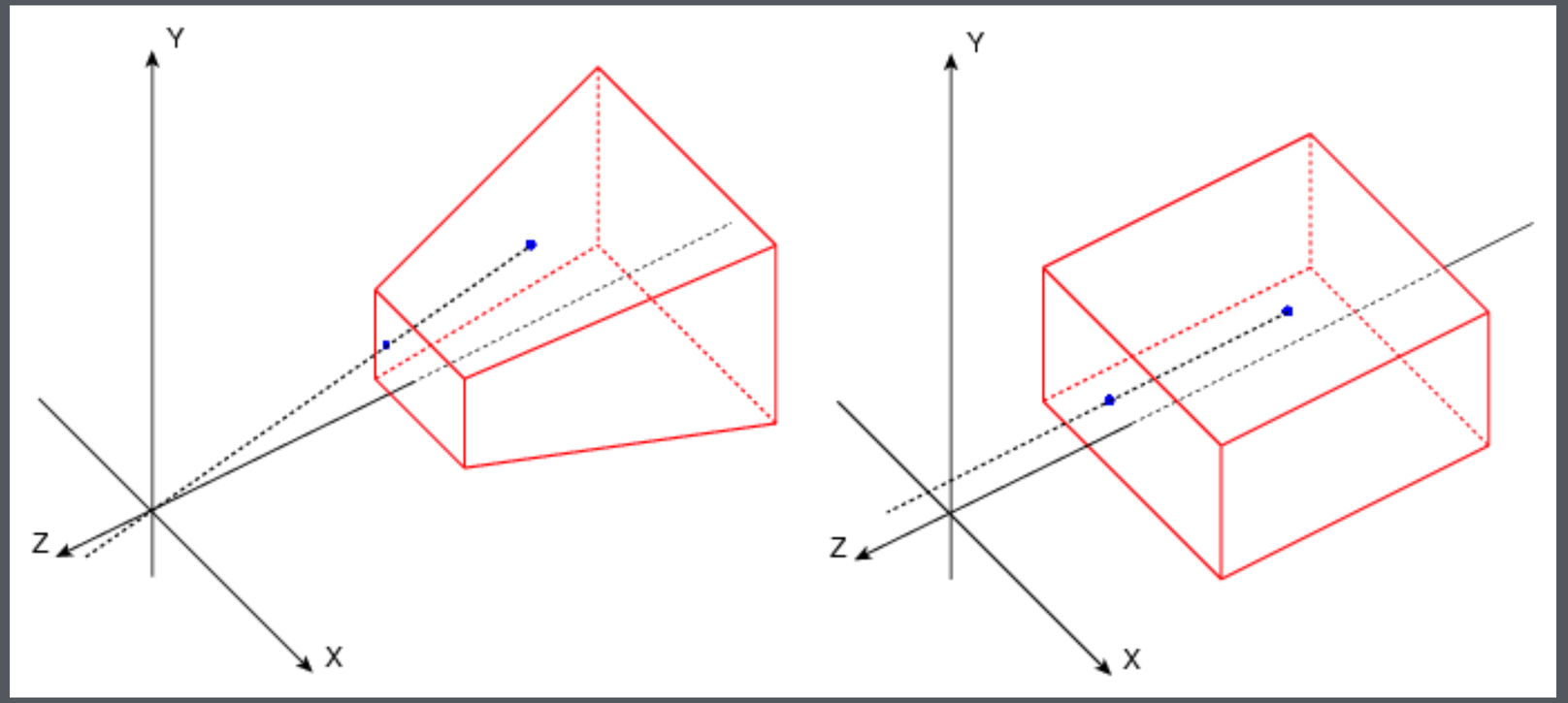


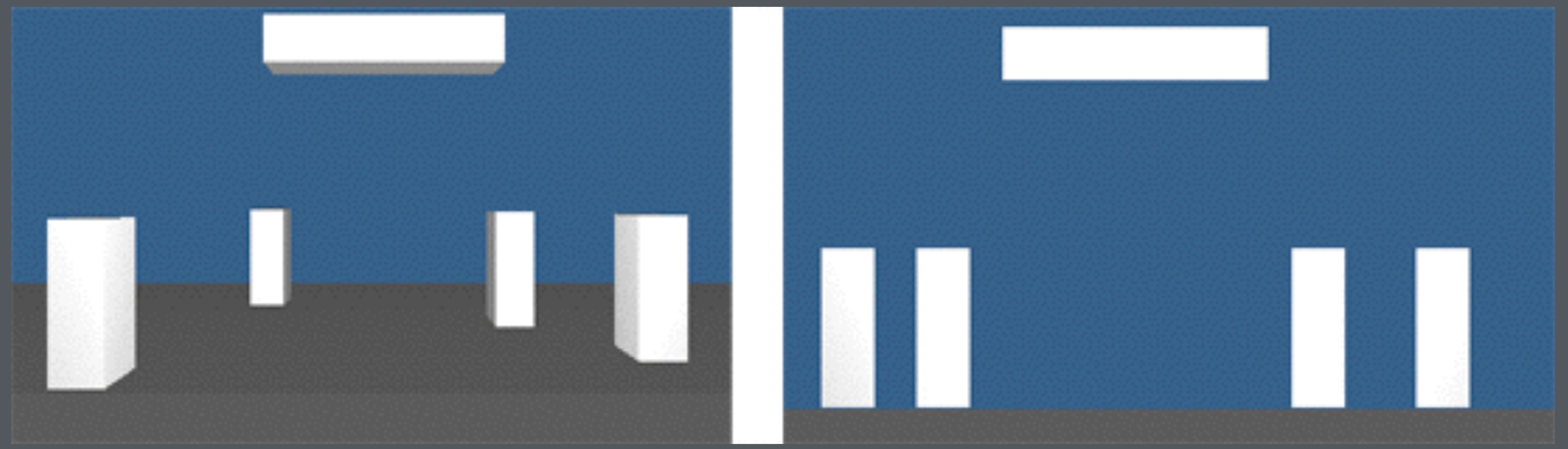
Perspective projection (P)



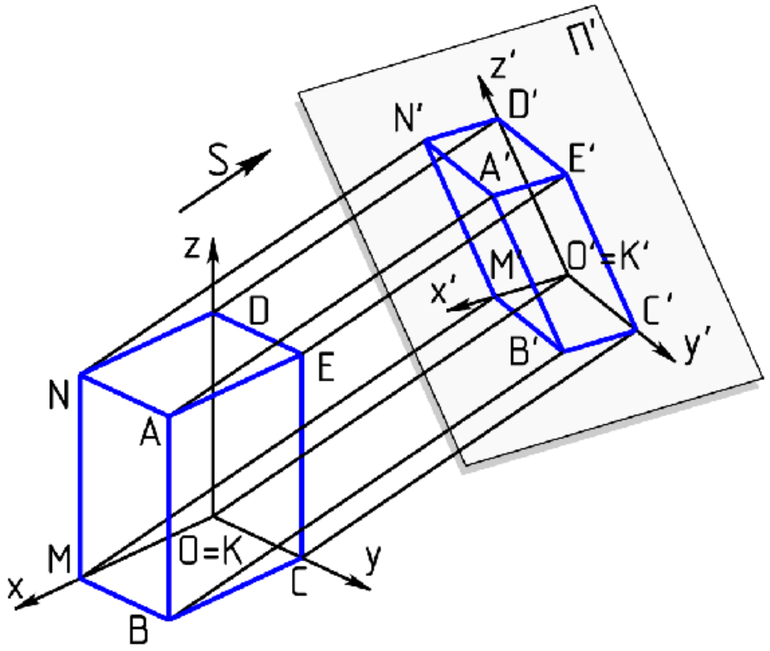
Orthographic projection (O)

Perspective vs. **Orthographic** projection





Projection matrix

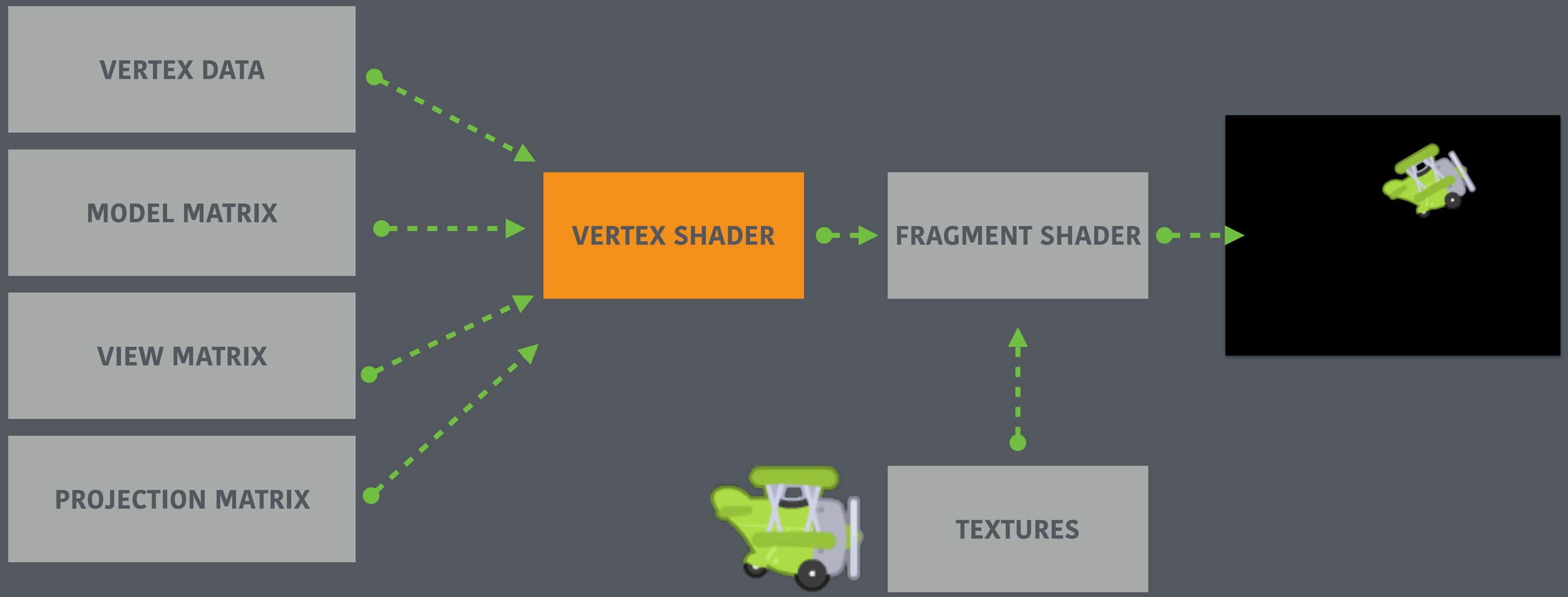


Vertex position * projection matrix = screen vertex position in normalized device coordinates

The projection matrix is a transformation matrix that is multiplied with all vertices at the end 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

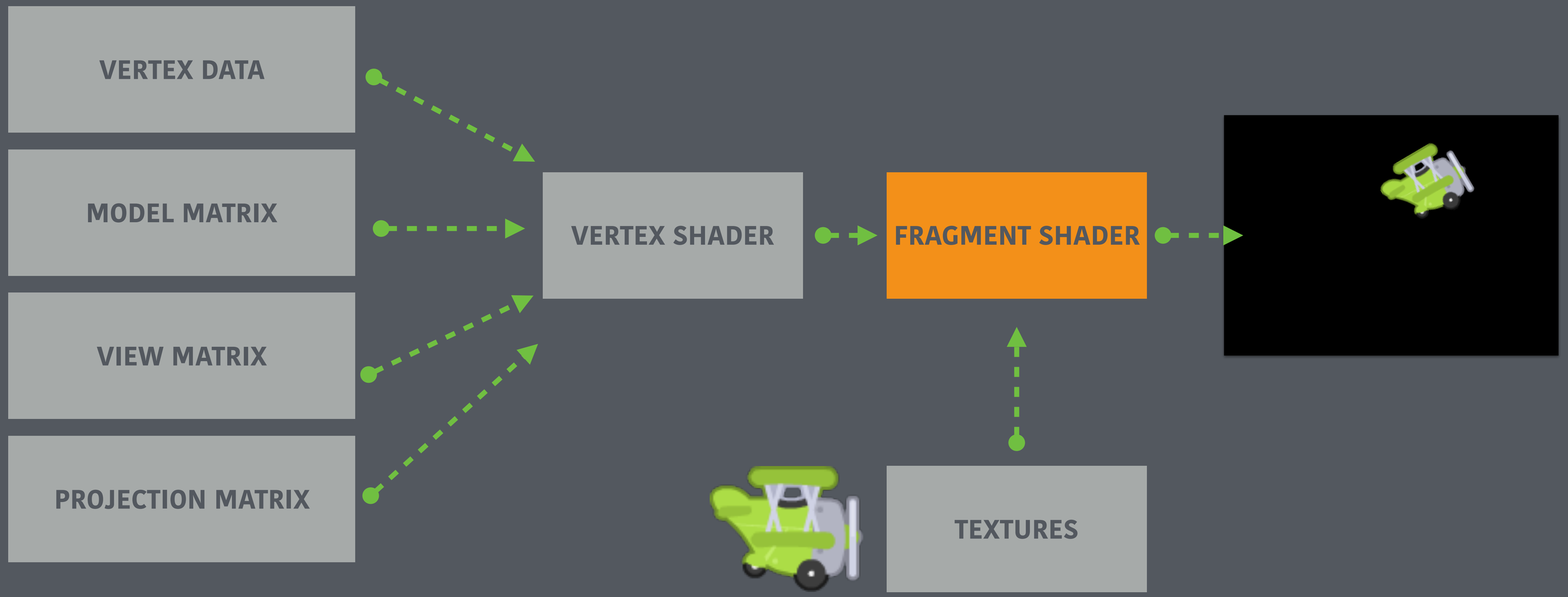


The vertex shader

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

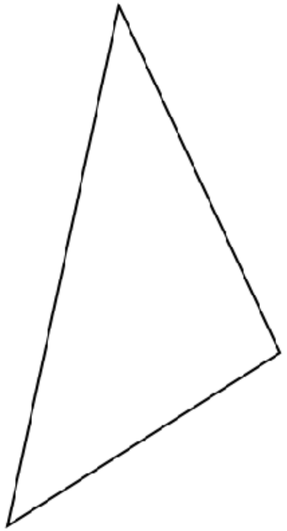
This is where our vertices are multiplied with our
transformation matrices.

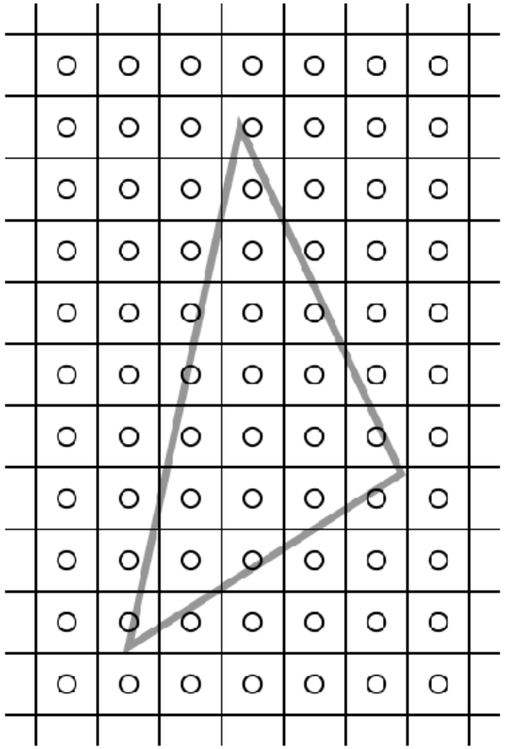
The **GPU** pipeline

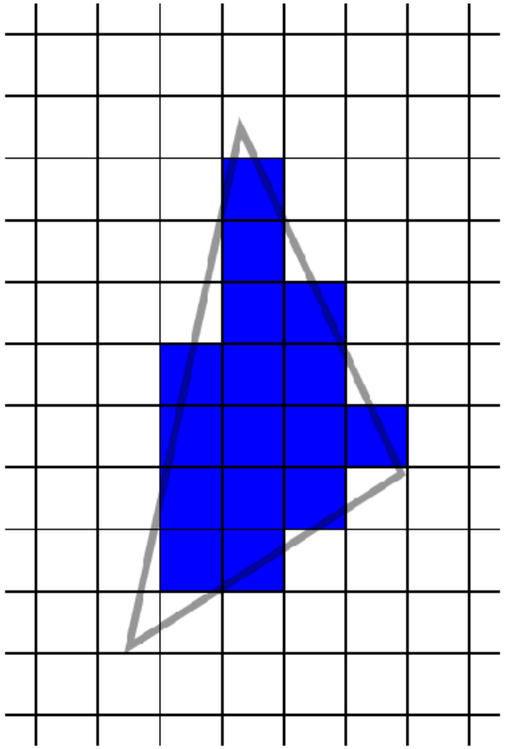


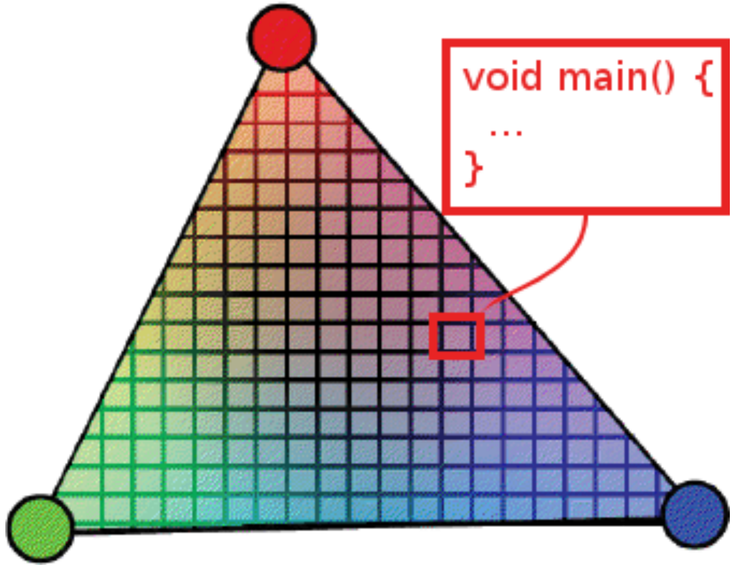
The fragment shader

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

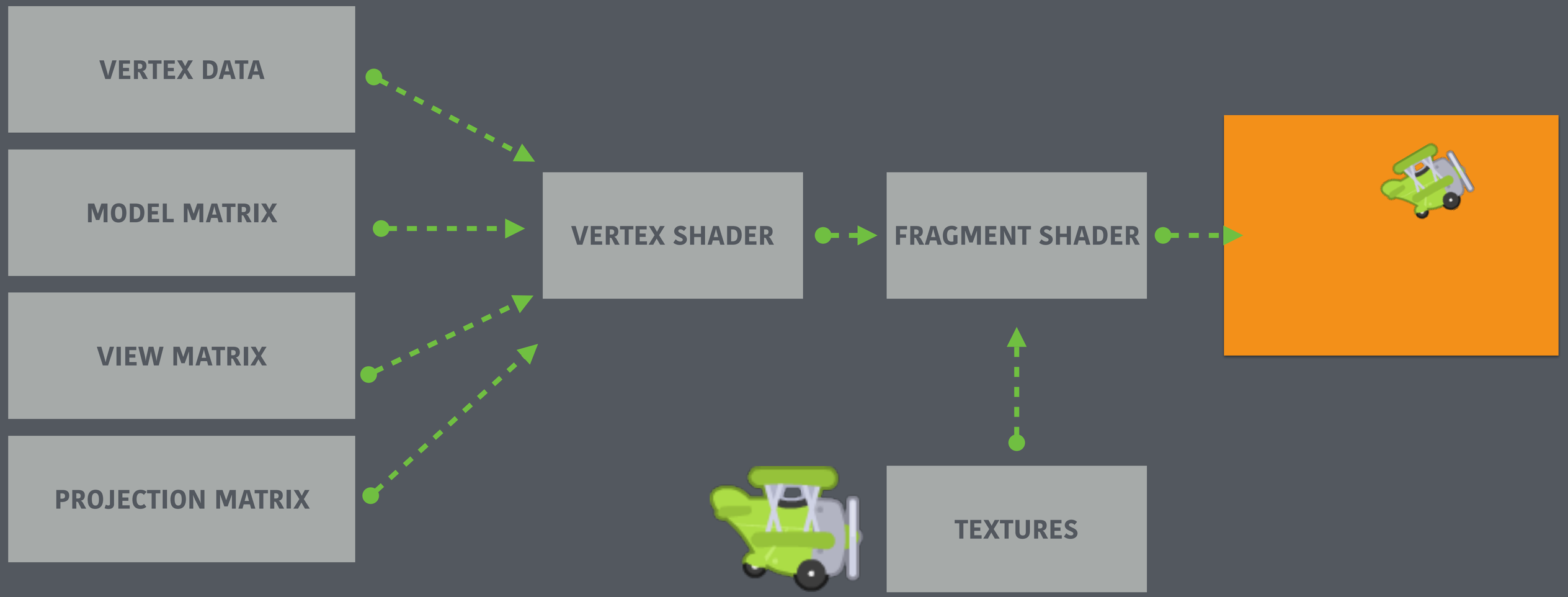


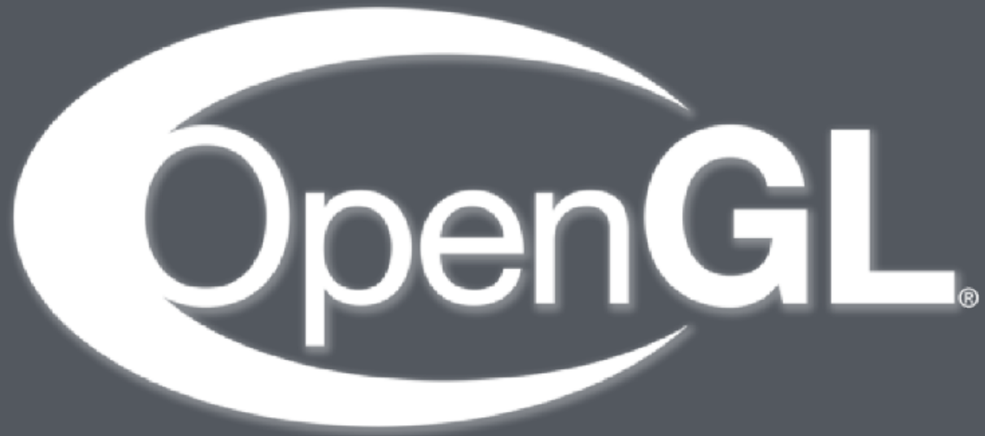





void main() {
...
}

The **GPU** pipeline




OpenGL®

The setup
(happens only once!)

VERTEX DATA

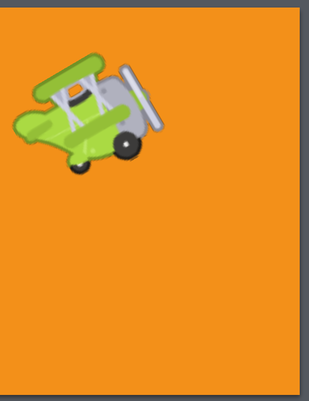
MODEL MATRIX

VIEW MATRIX

PROJECTION MATRIX

VERTEX SHADER

FRAGMENT SHADER

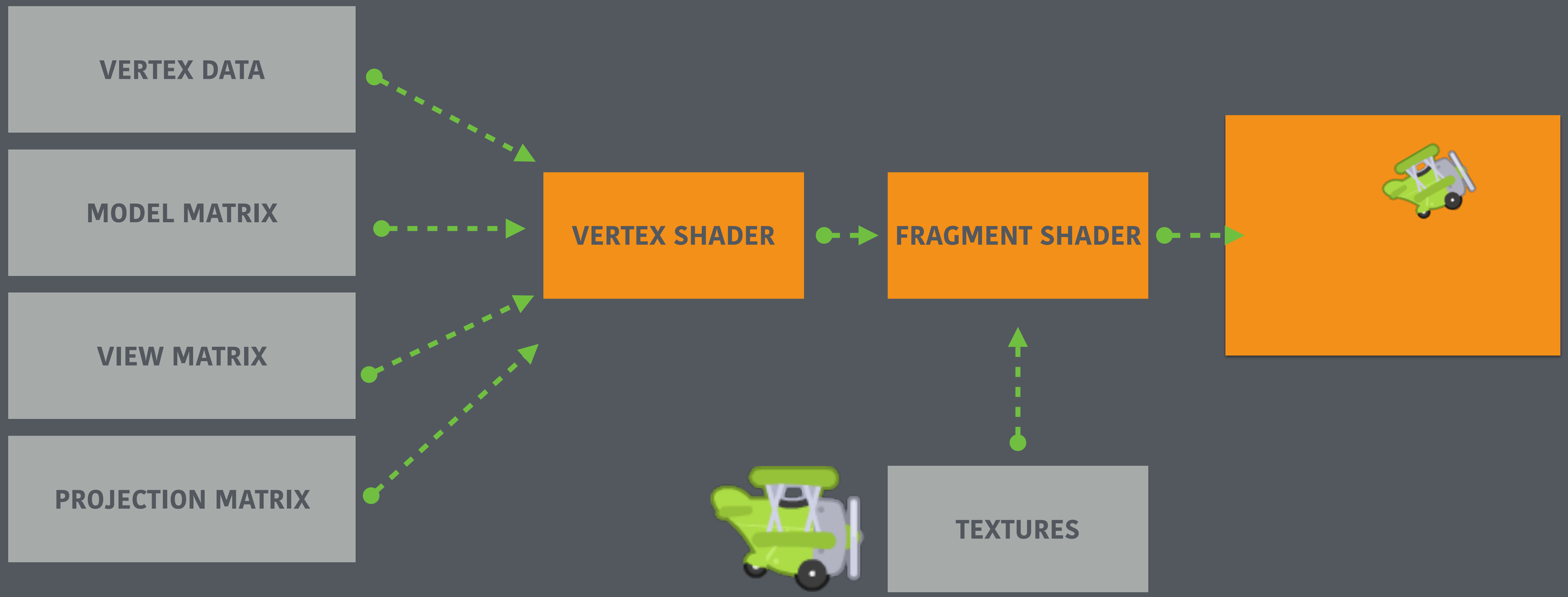


TEXTURES

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

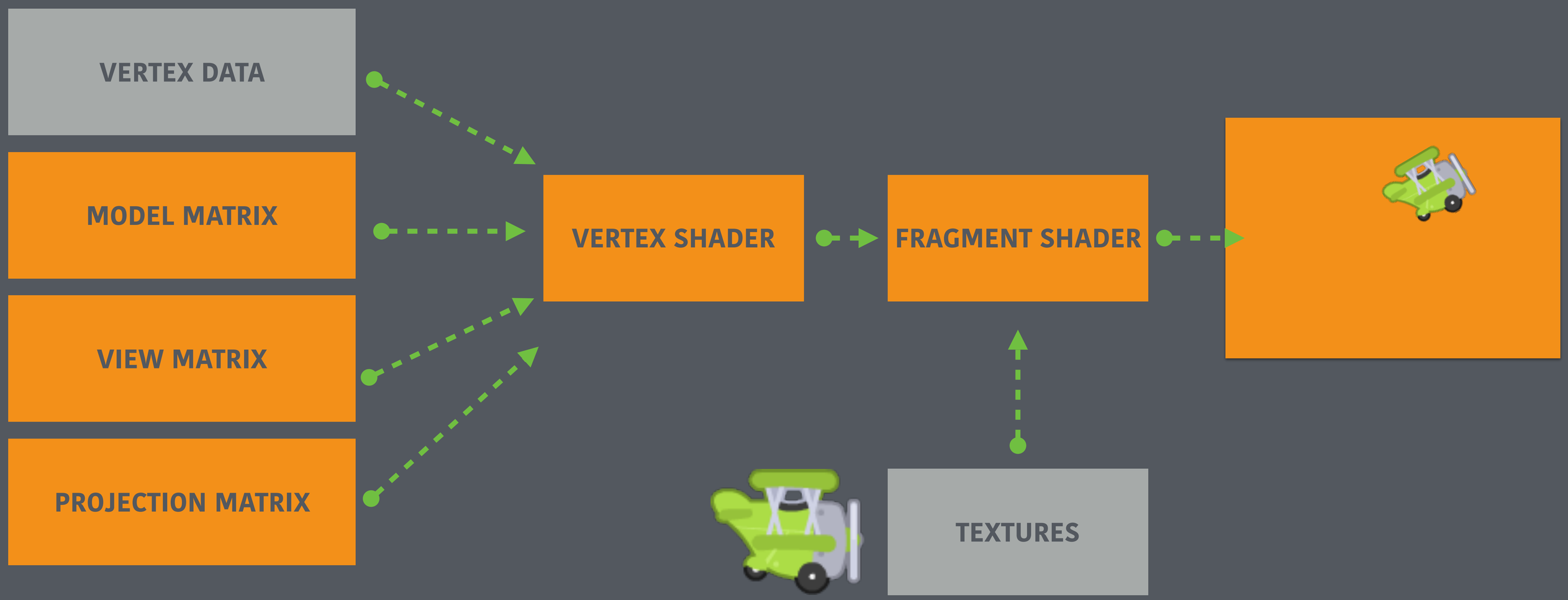
Sets the size and offset of rendering area (in pixels).

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


VERTEX DATA
MODEL MATRIX
VIEW MATRIX
PROJECTION MATRIX
VERTEX SHADER
FRAGMENT SHADER
TEXTURES

The **ShaderProgram** class.

```
#include "ShaderProgram.h"

// …

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


VERTEX DATA
MODEL MATRIX
VIEW MATRIX
PROJECTION MATRIX
VERTEX SHADER
FRAGMENT SHADER
TEXTURES

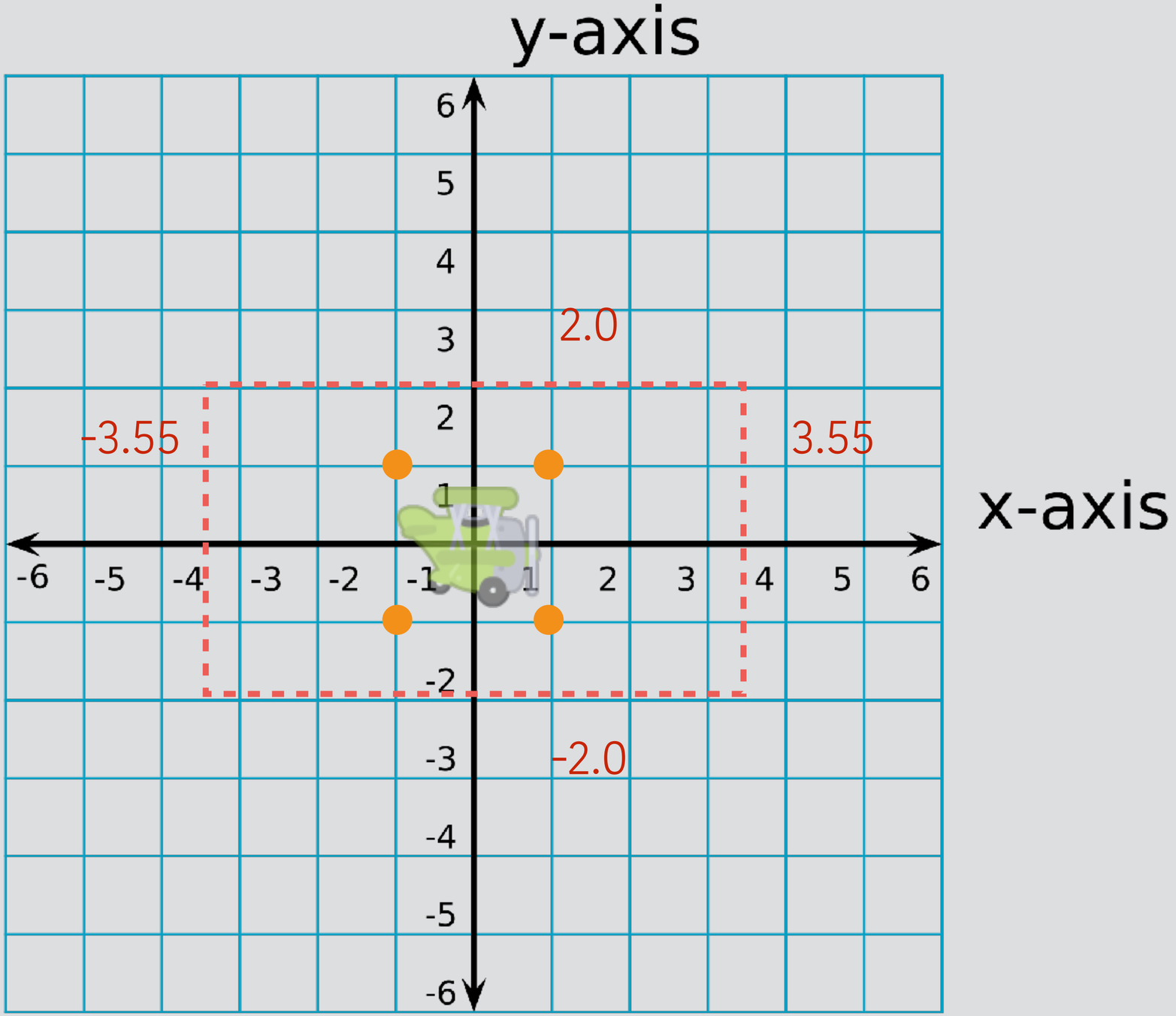
The **Matrix** class.

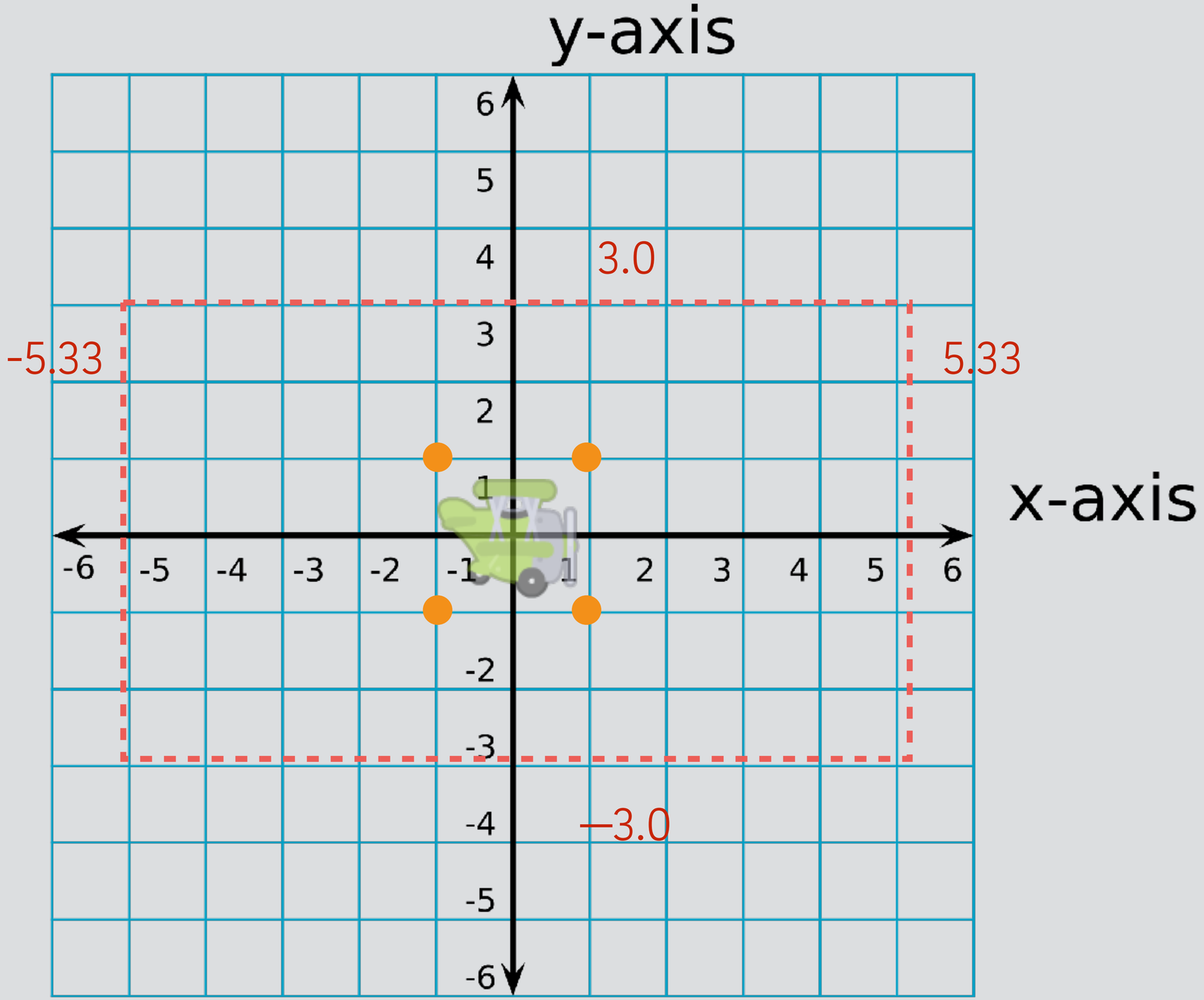
```
#include "Matrix.h"

// …

Matrix projectionMatrix;
Matrix modelMatrix;
Matrix viewMatrix;
```

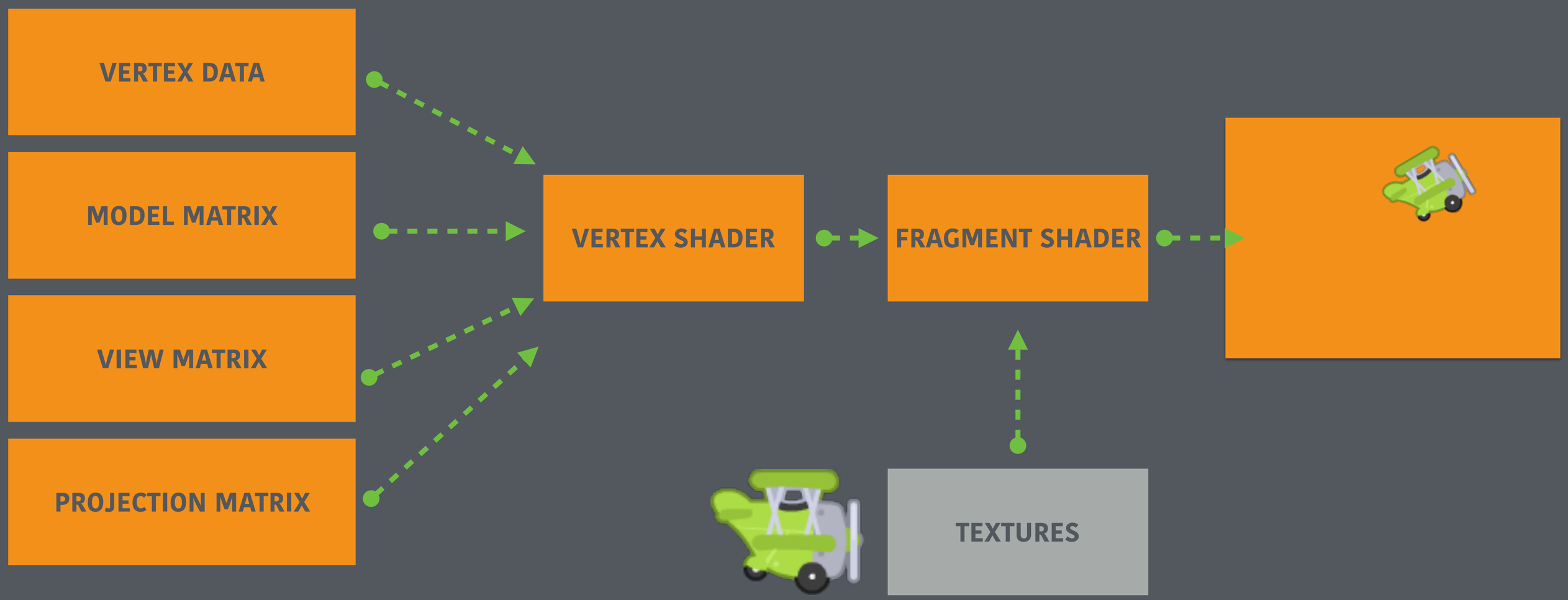
```
void Matrix::SetOrthoProjection (float left, float right,
float bottom, float top, float near, float far);
```

Sets an orthographic projection in a matrix.

```
Matrix projectionMatrix;
projectionMatrix.SetOrthoProjection(-3.55, 3.55, -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


VERTEX DATA
MODEL MATRIX
VIEW MATRIX
PROJECTION MATRIX
VERTEX SHADER
FRAGMENT SHADER
TEXTURES

Drawing polygons.
(happens every frame!)

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

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

```
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);
```

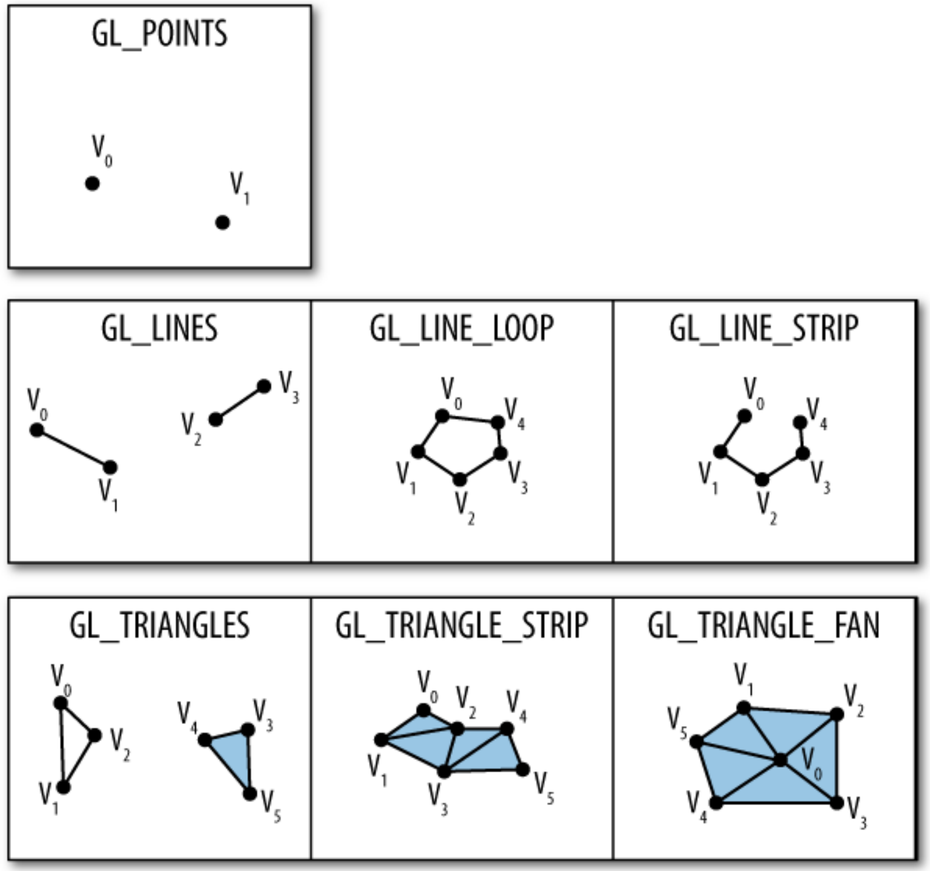
Enables a **vertex attribute**.

```
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)

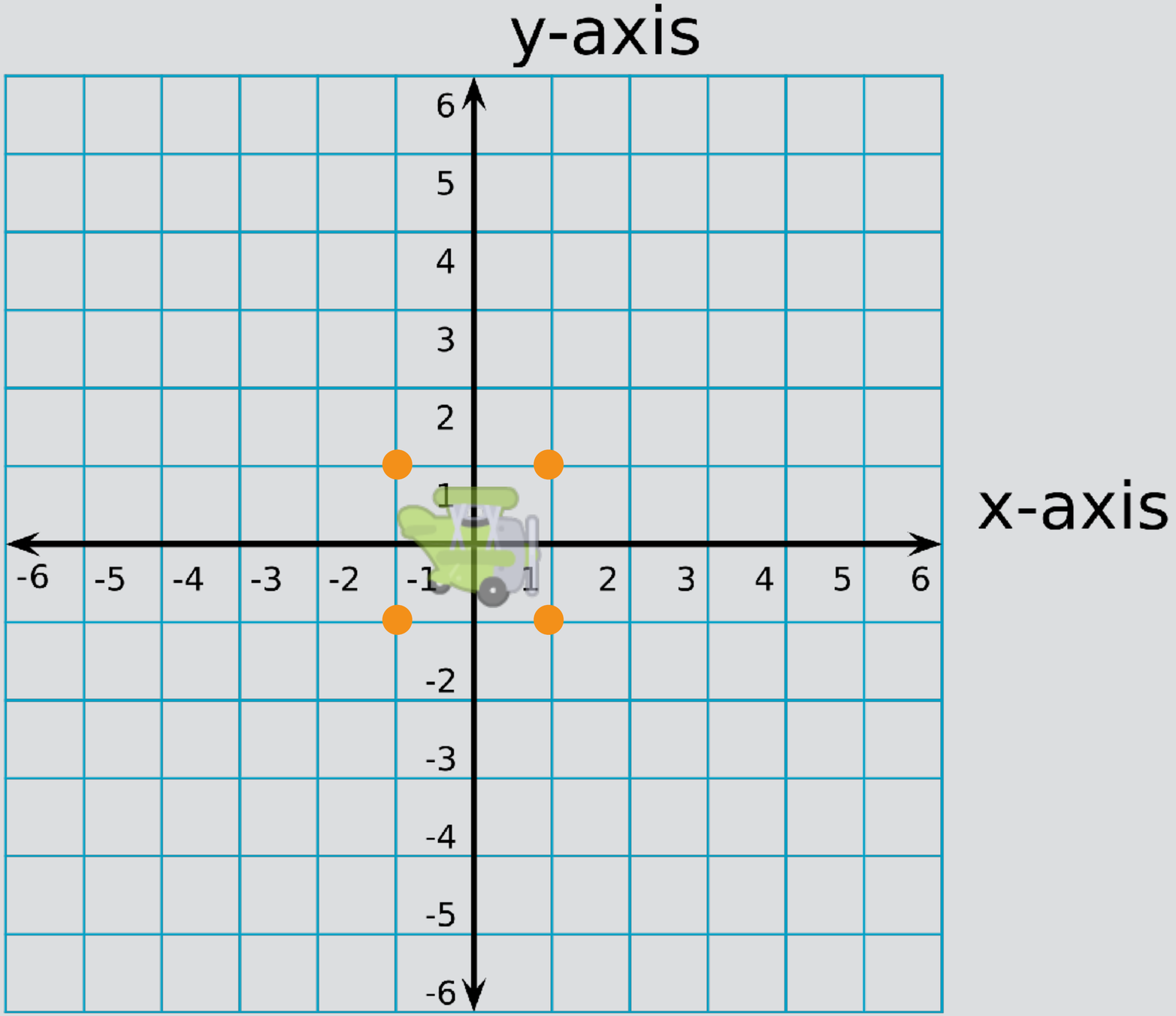
```
glViewport(0, 0, 640, 360);

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

Matrix projectionMatrix;
Matrix modelMatrix;
Matrix viewMatrix;

projectionMatrix.SetOrthoProjection(-3.55, 3.55, -2.0f, 2.0f, -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.

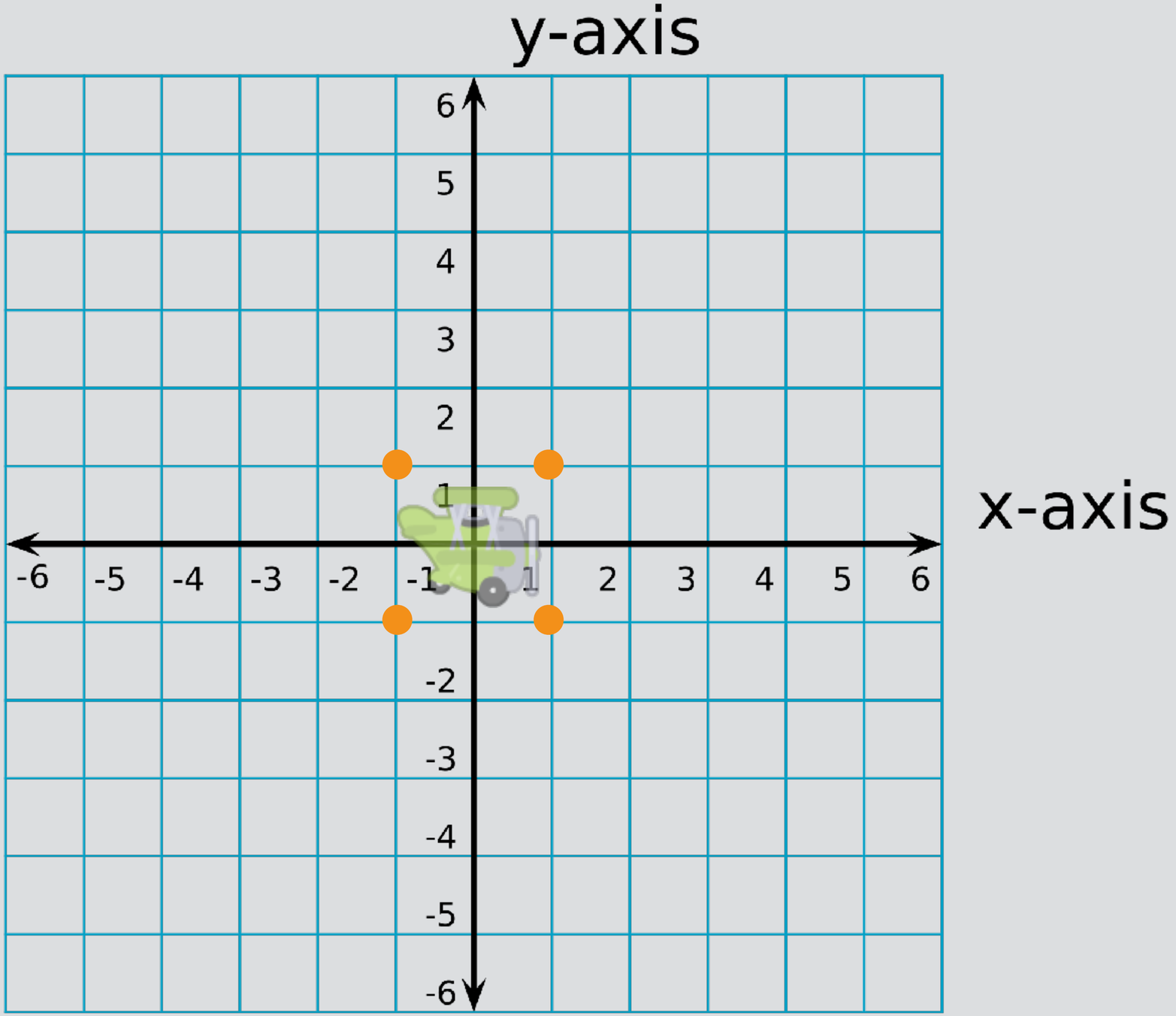

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

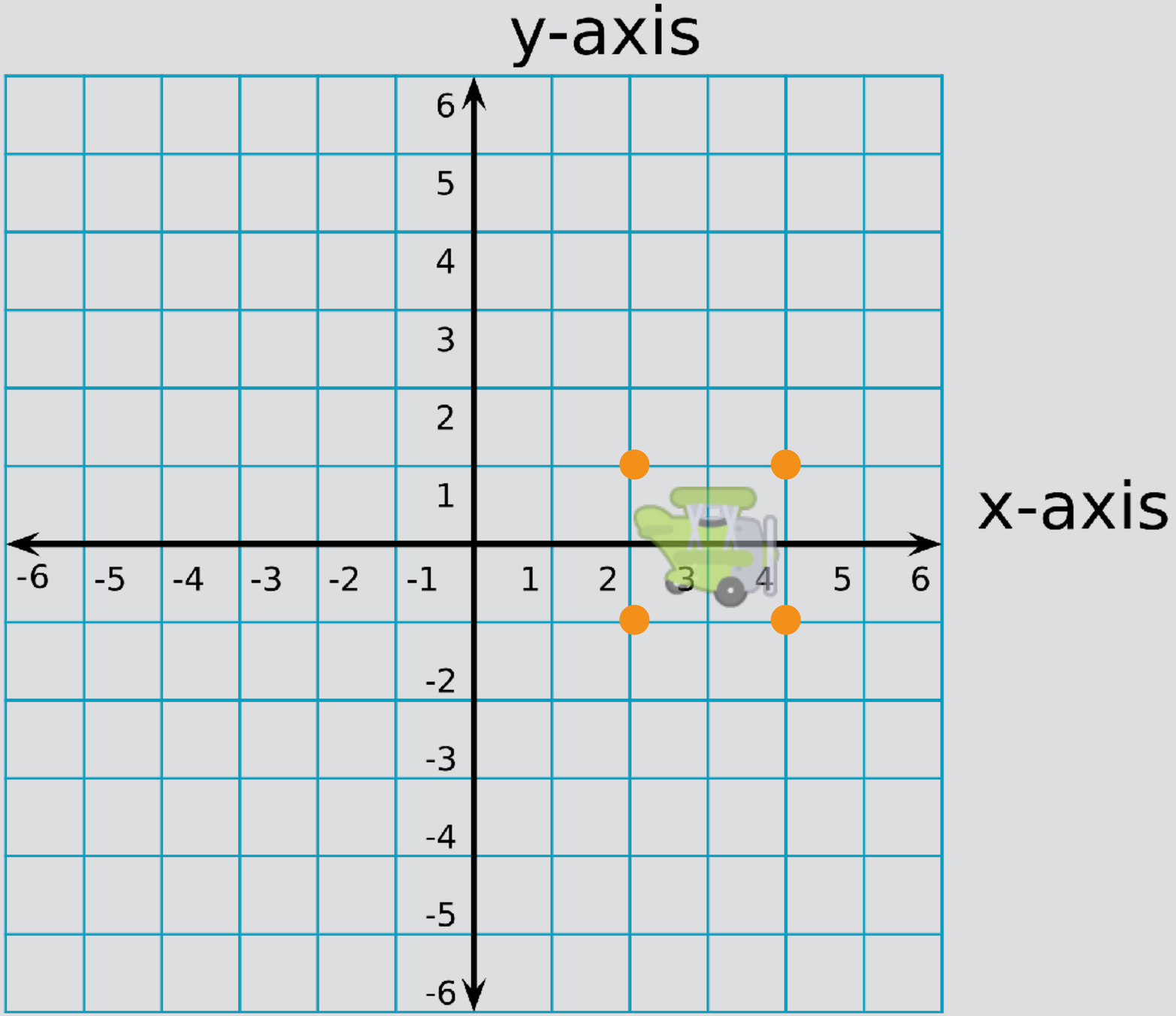
```
void Matrix::identity();
```

Resets a matrix to have no transformations.

```
Matrix modelMatrix;
modelMatrix.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

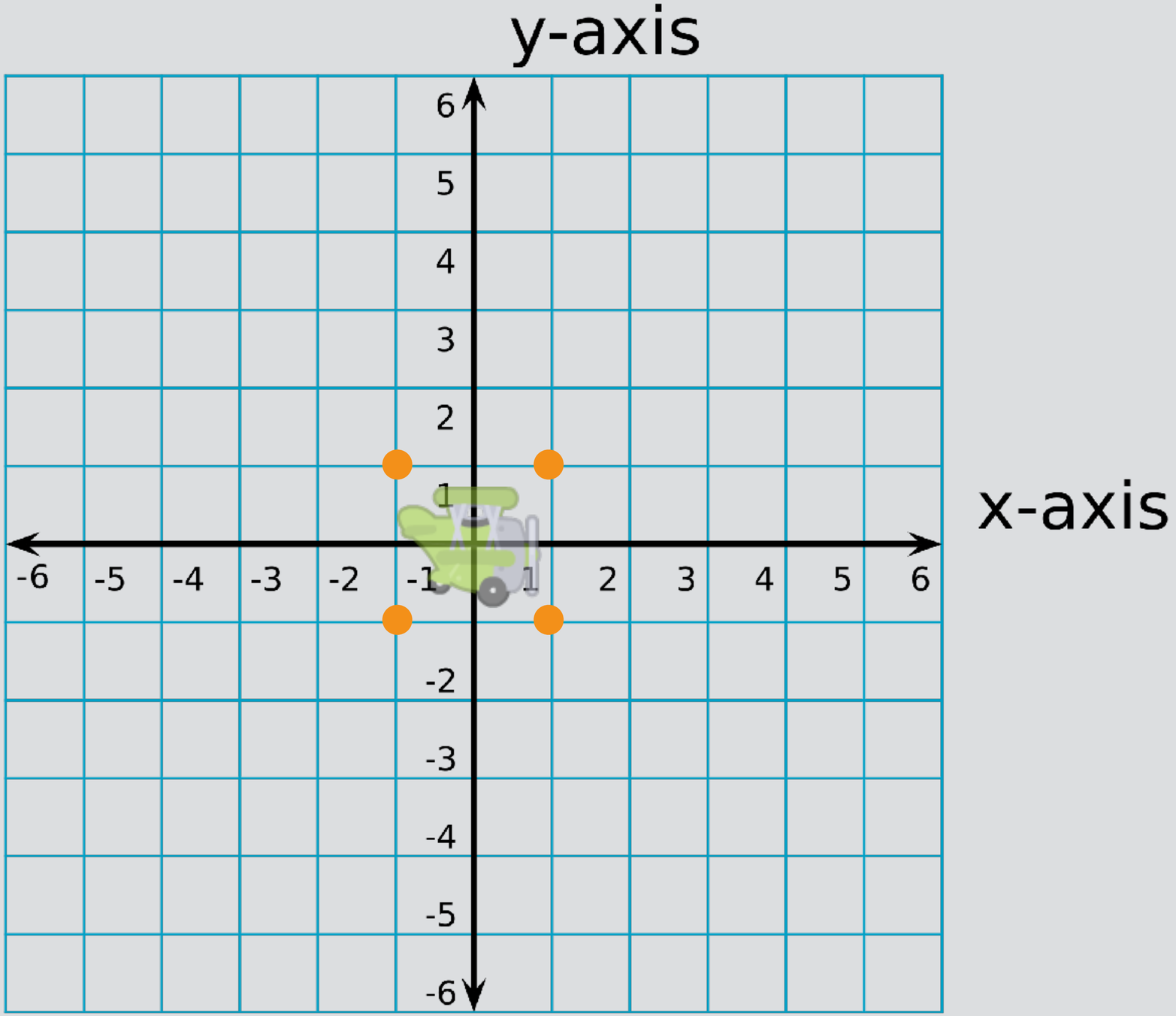

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

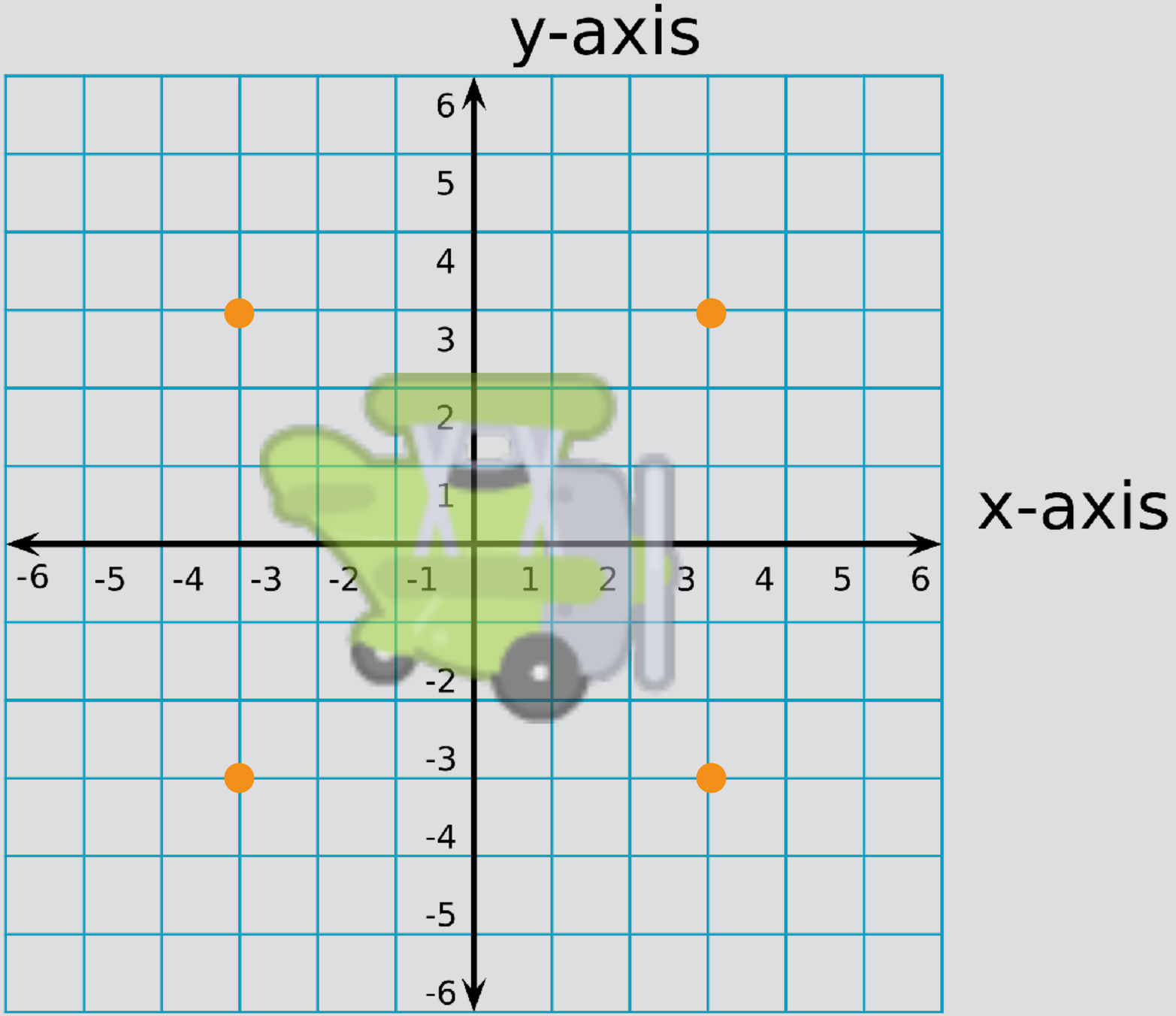
```
void Matrix::Translate (float x, float y, float z);
```

Translates a matrix by specified coordinates.

```
Matrix modelMatrix;
modelMatrix.Translate(1.0f, 0.0f, 0.0f);
```

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

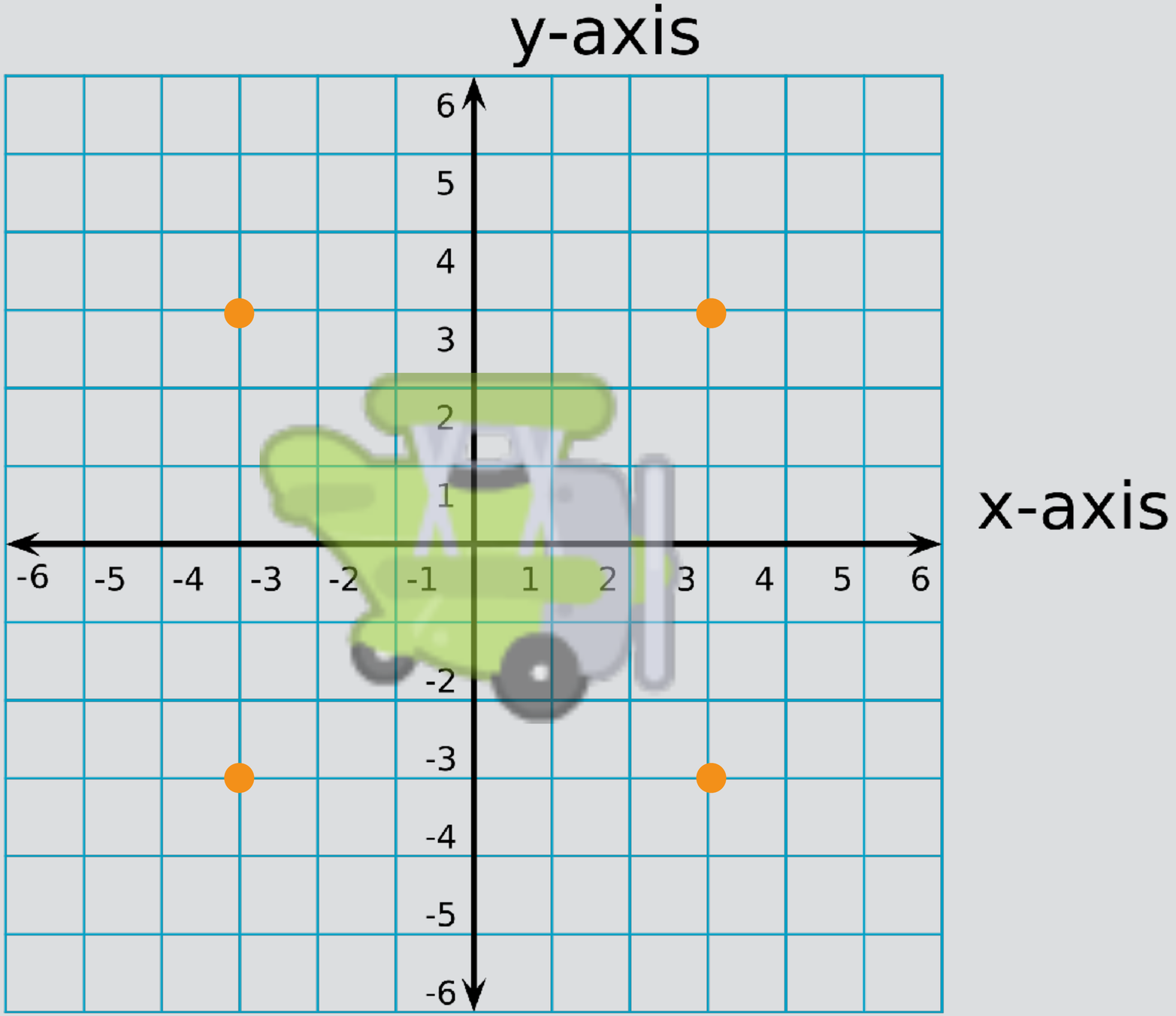

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

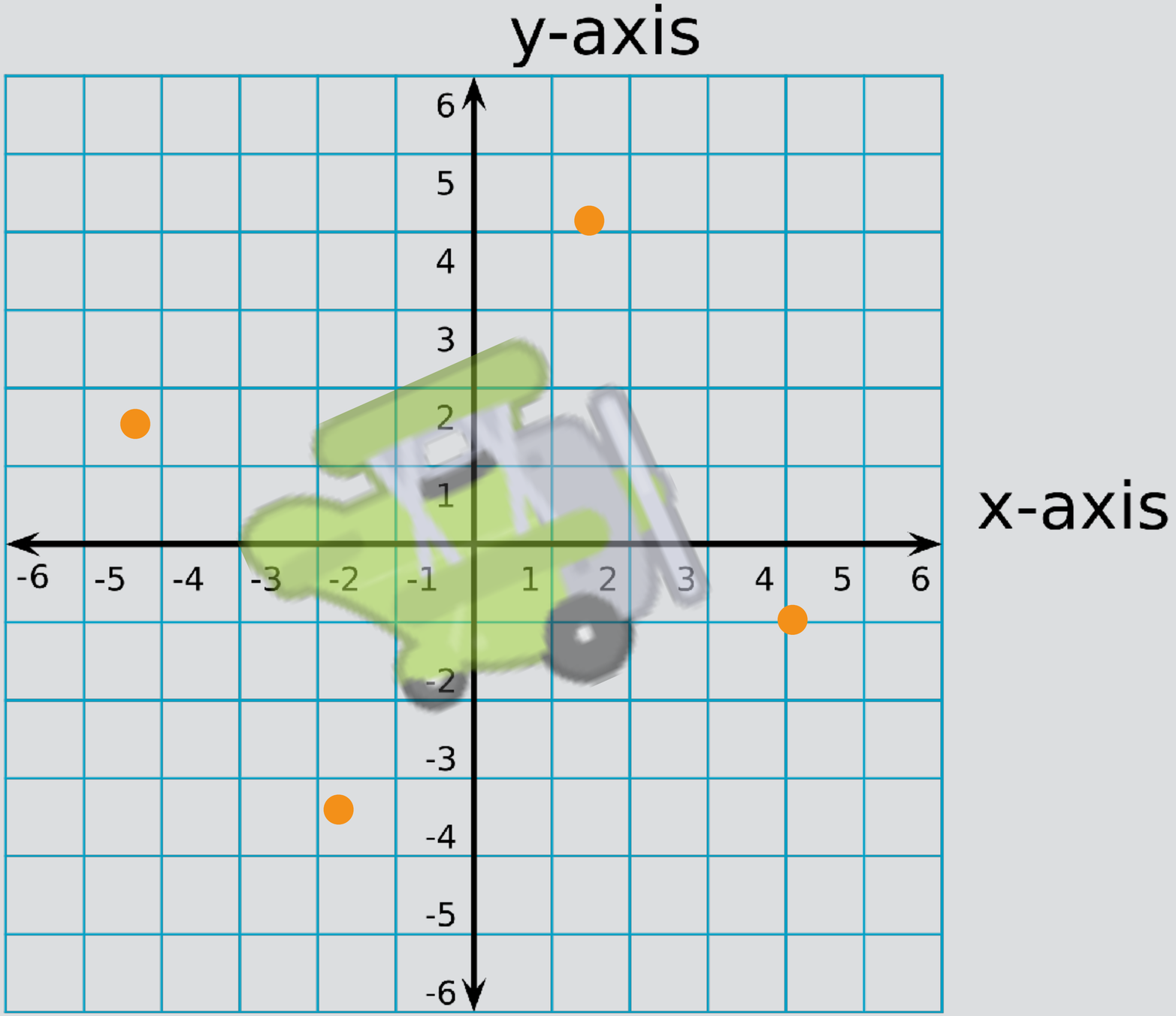
```
void Matrix::Scale (float x, float y, float z);
```

Scales a matrix by specified coordinates.

```
Matrix modelMatrix;
modelMatrix.Scale(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

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

Rotates a matrix by specified radians.

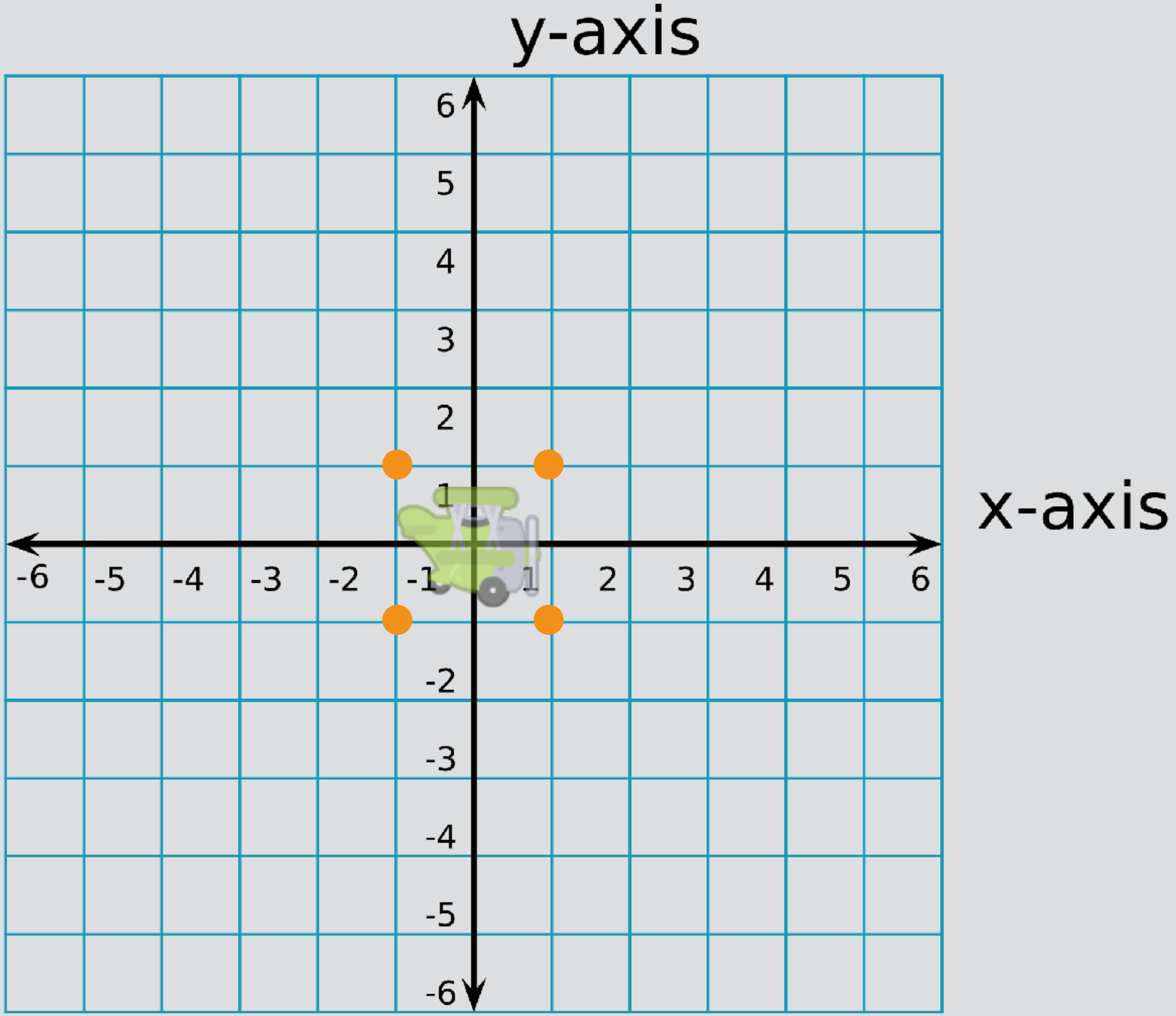
```
Matrix modelMatrix;
modelMatrix.Rotate(45.0f * (3.1415926f / 180.0f));
```

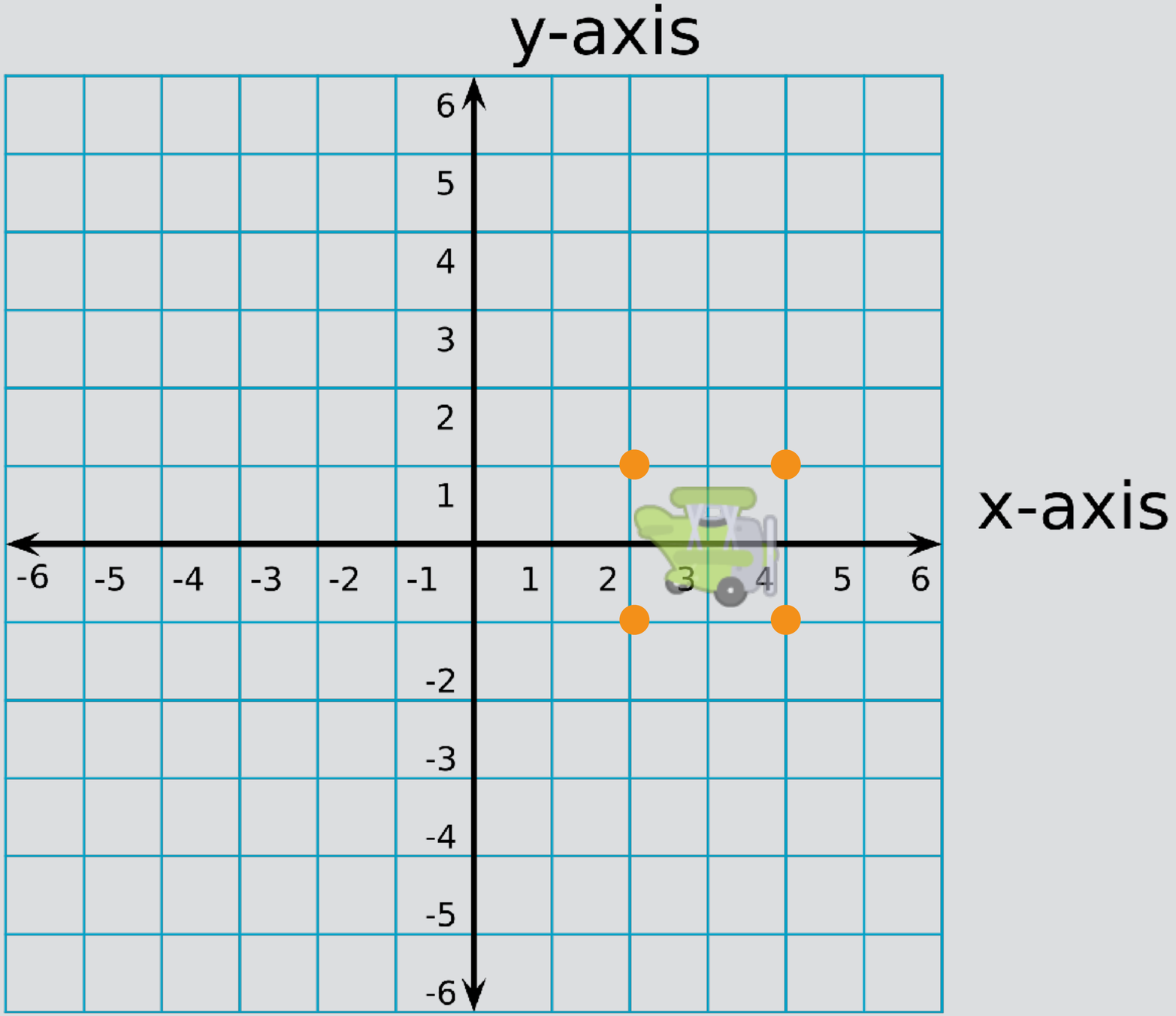
To convert from **degrees** to **radians**:

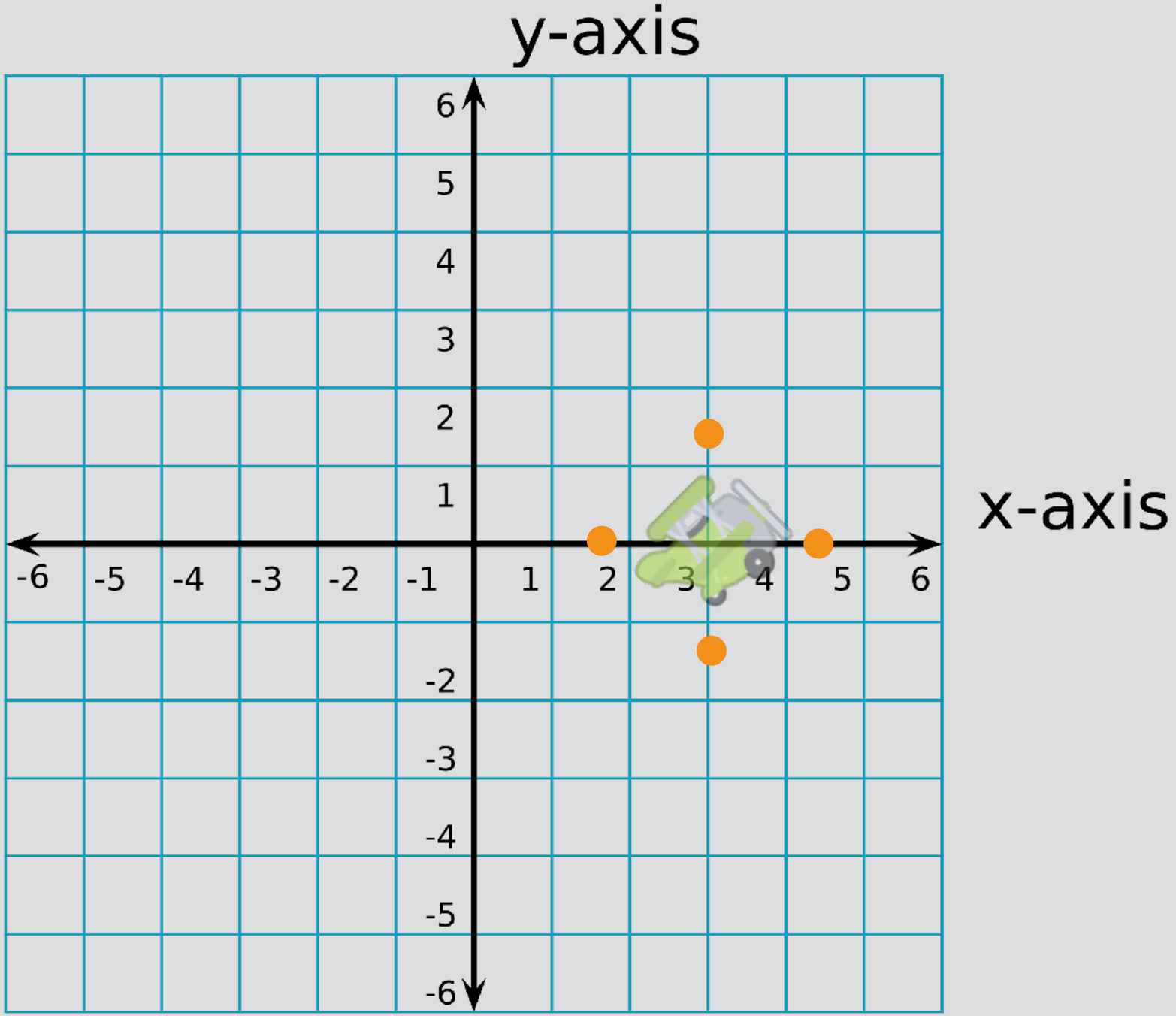
RADIANS = DEGREES * (PI / 180)

Matrix transformations
are not commutative!

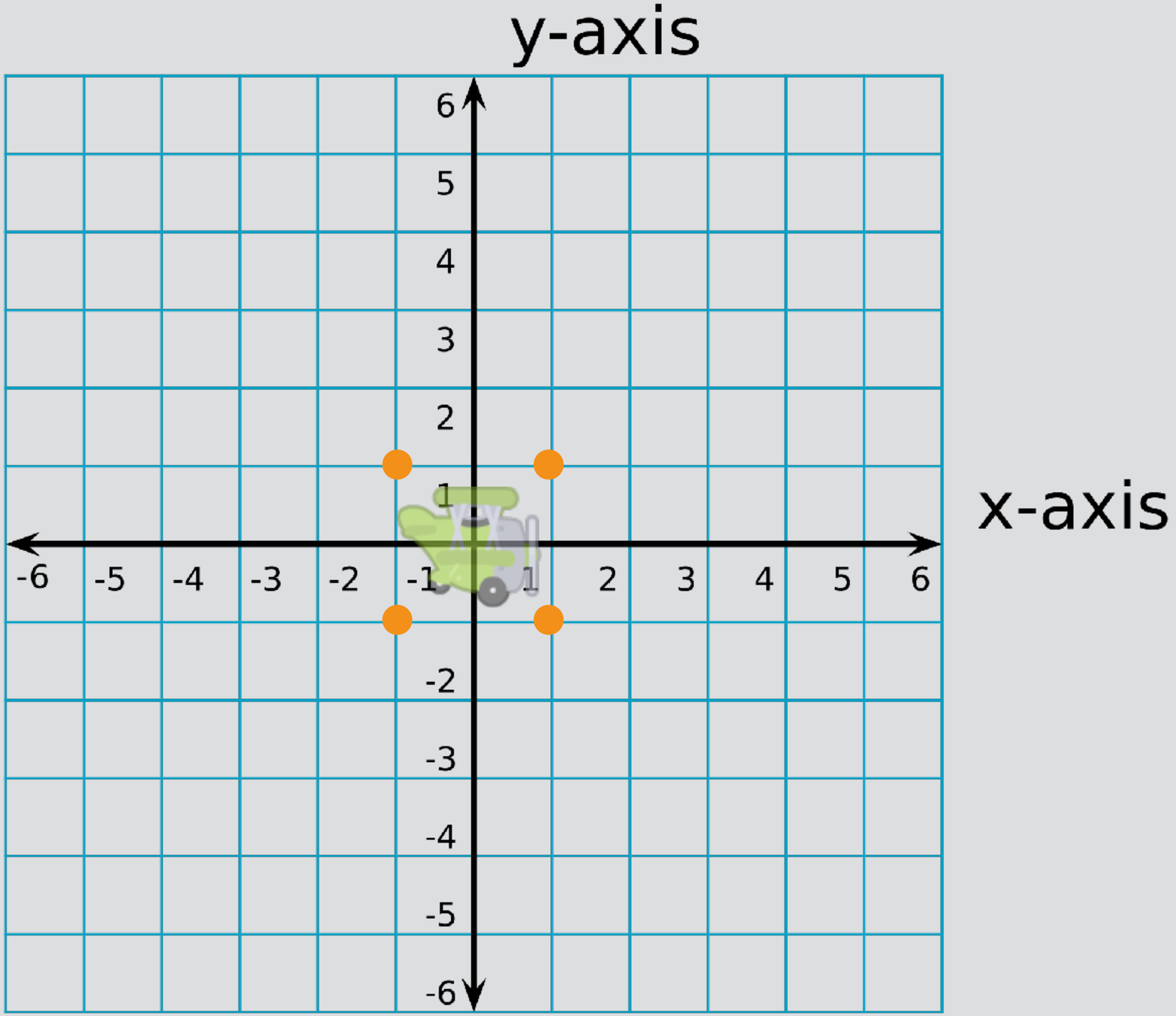
```
Matrix modelMatrix;
modelMatrix.Translate(2.0f, 0.0f, 0.0f);
modelMatrix.Rotate(45.0f * (3.1415926f / 180.0f));
```

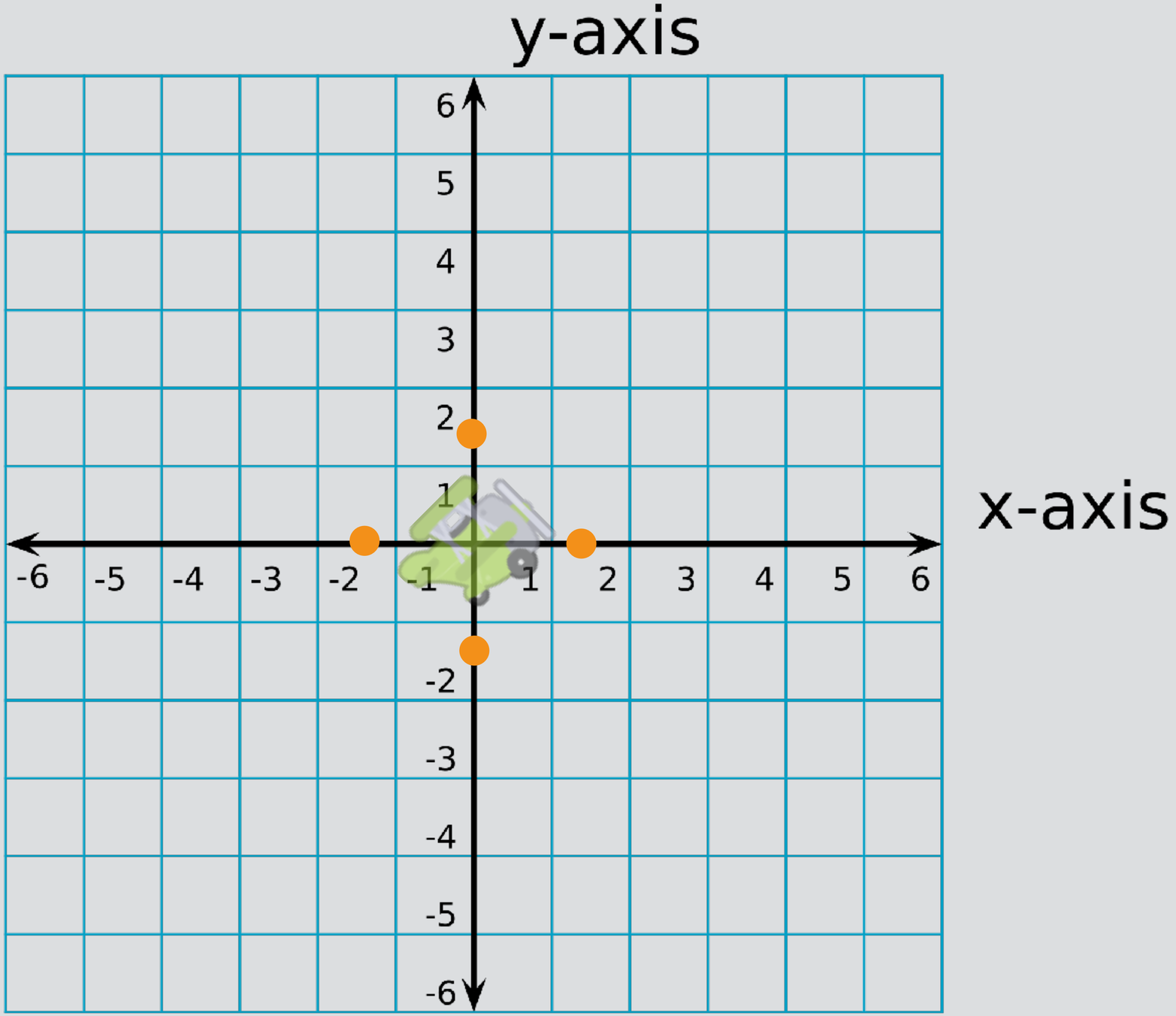

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

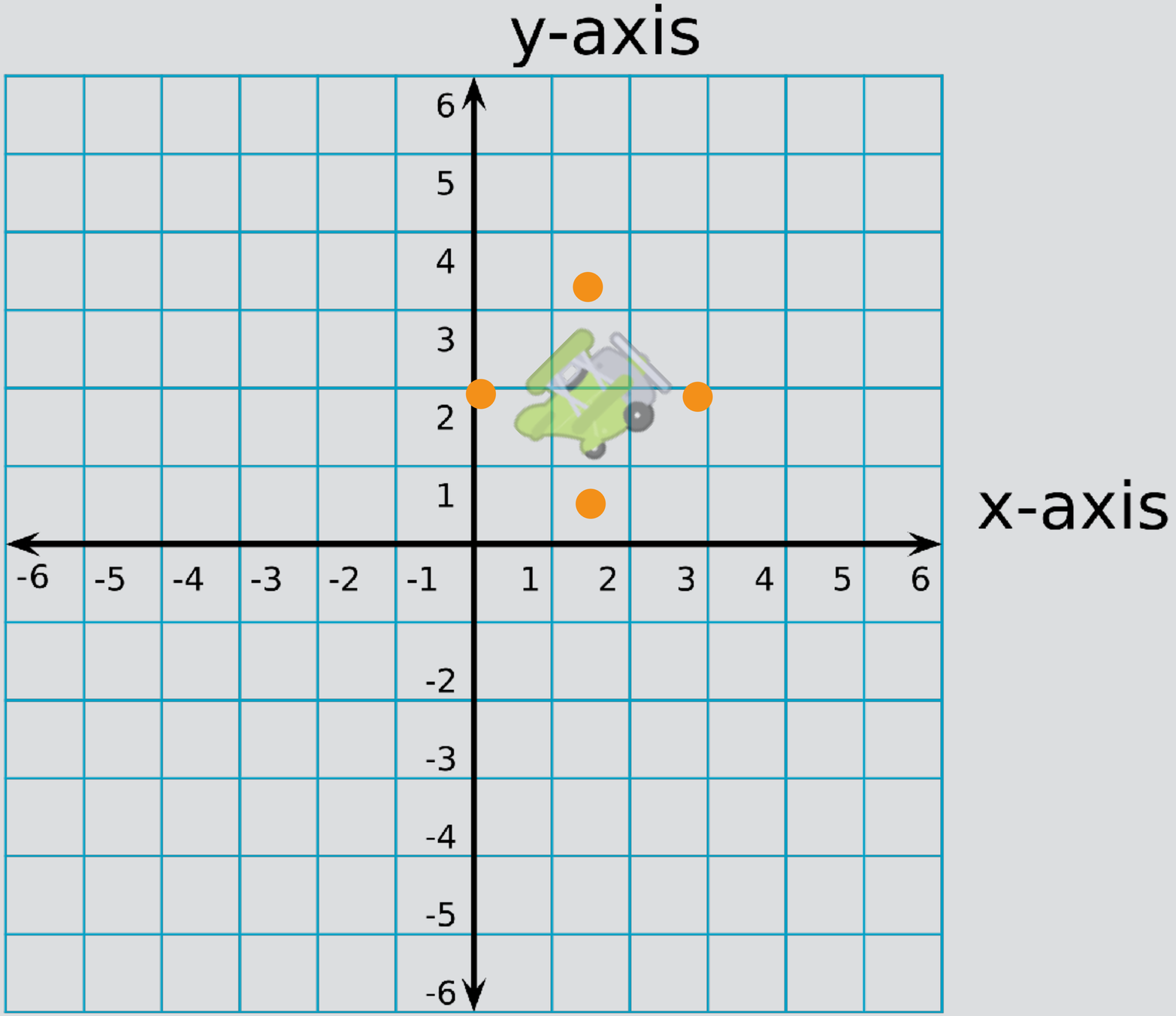

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

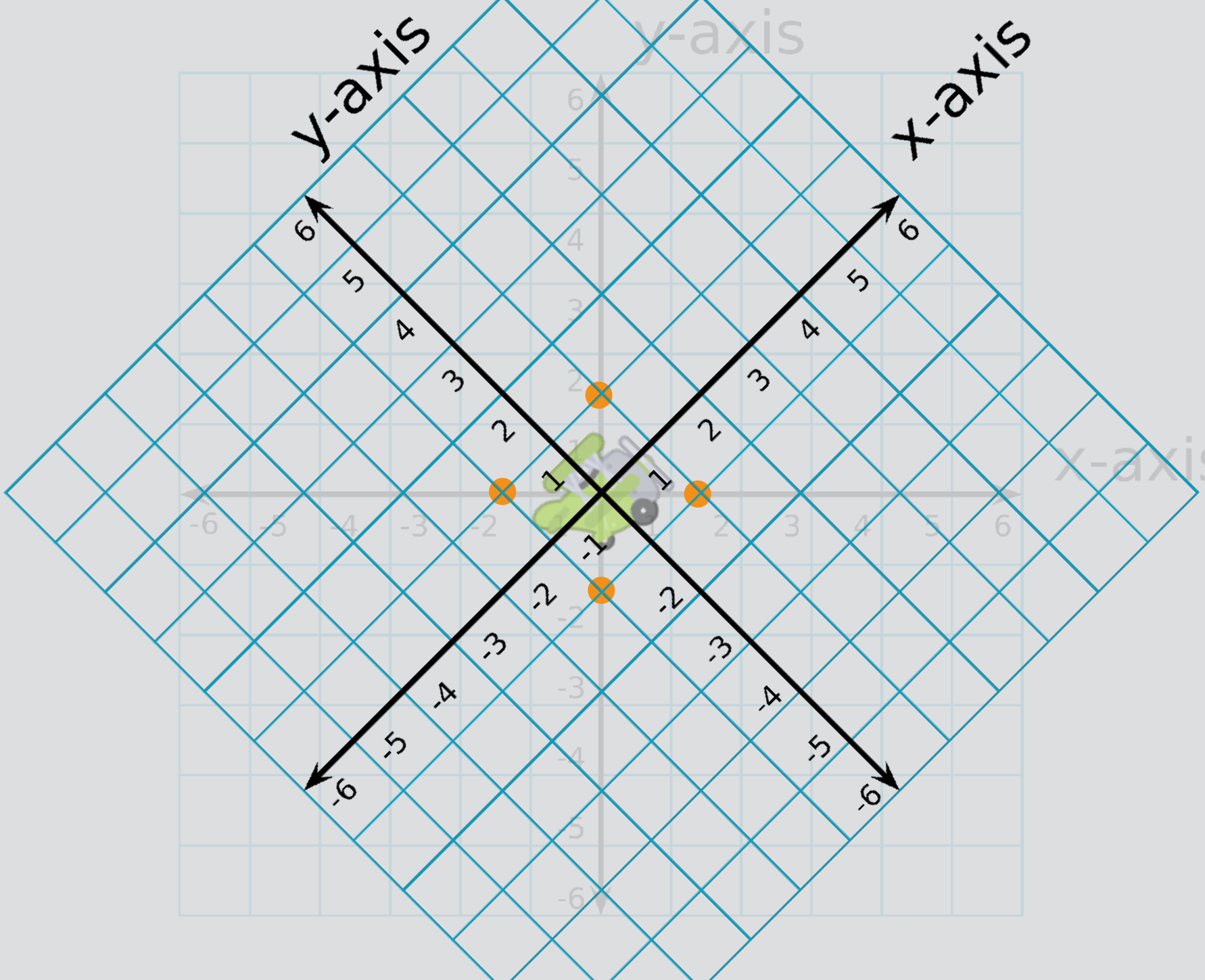

y-axis
x-axis
6
5
4
3
2
1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6

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

Multiple transformations.

```
modelMatrix.Translate(2.0f, 0.0f, 0.0f);
program.SetModelMatrix(modelMatrix);

// draw first object

modelMatrix.Identity();
modelMatrix.Translate(-2.0f, 0.0f, 0.0f);
program.SetModelMatrix(modelMatrix);

// draw second object
```

```
Matrix modelMatrix1;
modelMatrix1.Translate(2.0f, 0.0f, 0.0f);
program.SetModelMatrix(modelMatrix1);

// draw first object

Matrix modelMatrix2;
modelMatrix2.Translate(-2.0f, 0.0f, 0.0f);
program.SetModelMatrix(modelMatrix2);

// draw second object
```