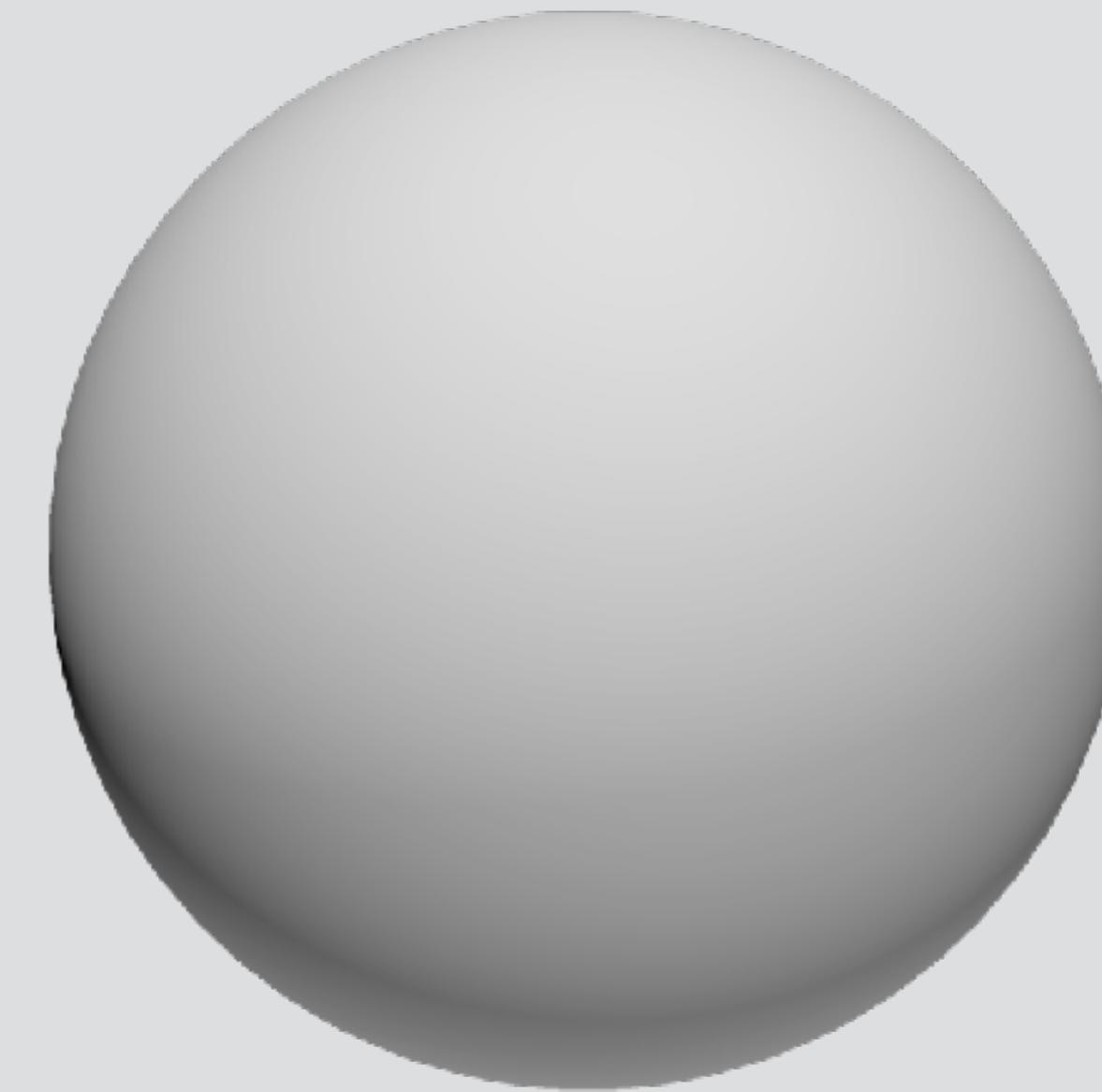
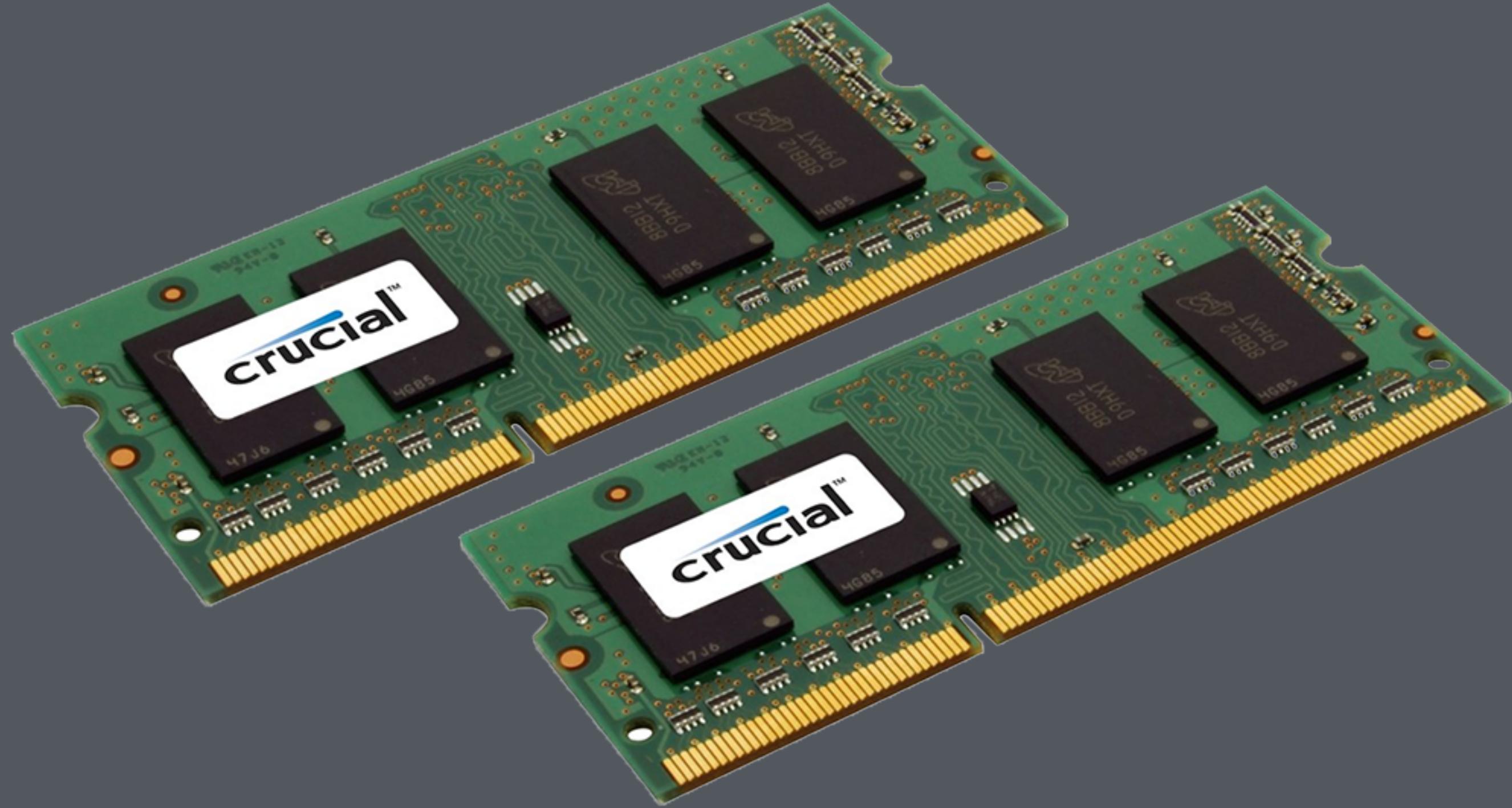
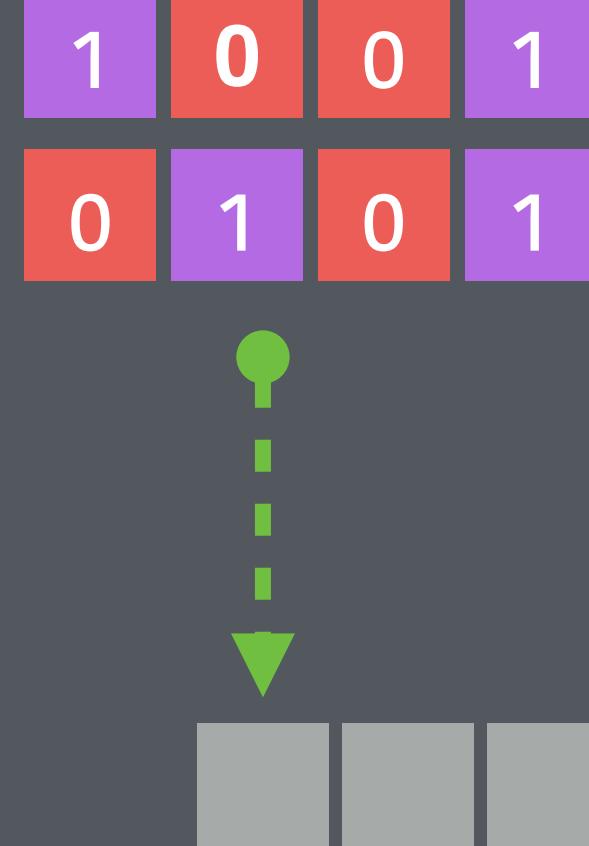


# Graphics Foundations



CS 3113

# Quick pointer refresher



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



0 1 2 3 4 5 6 7 8 9 ...

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



0 1 2 3 4 5 6 7 8 9 ...

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



0 1 2 3 4 5 6 7 8 9 ...

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



```
float *myVarPtr = &myVar;  
cout << myVarPtr << endl; // prints 3
```



```
float myVar = 16.0f;  
  
float *myVarPtr = &myVar; ←-----● Reference  
  
cout << myVarPtr << endl; // prints 3  
  
cout << *myVarPtr << endl; // prints 16 ←-----● Dereference  
  
cout << &myVar << endl; // prints 3
```



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

```
float *myVarPtr = &myVar; ←-----● Reference
```

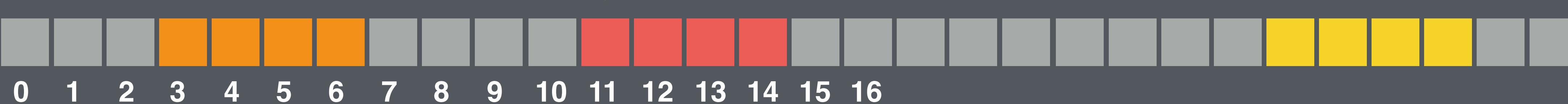
```
cout << myVarPtr; // prints 3
```

```
cout << *myVarPtr; // prints 16 ←-----● Dereference
```

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

```
float **myVarPtrPtr = &myVarPtr;
```

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

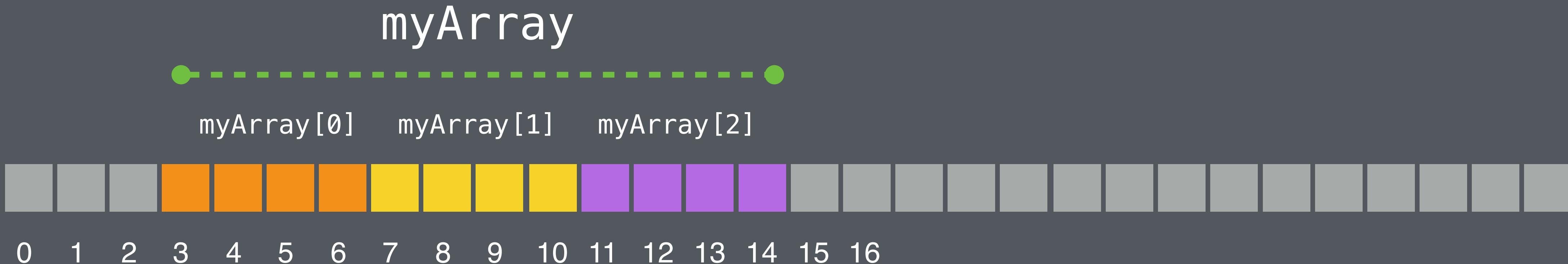


```
float myArray[3] = {13.5f, 2.3f, 5.4f};

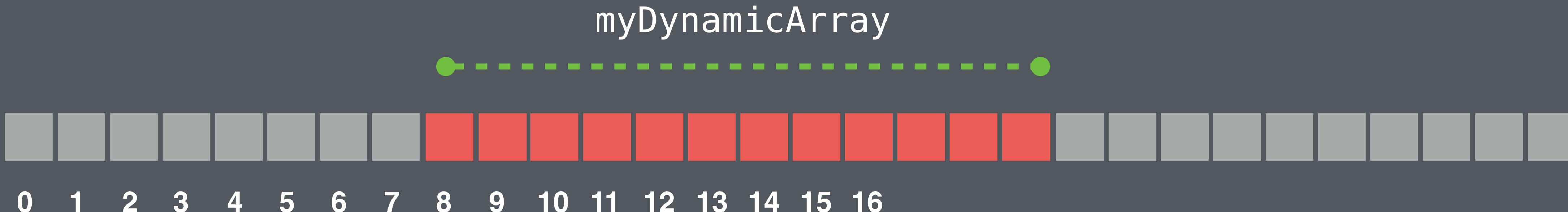
cout << myArray[0]; // prints 13.5
cout << myArray; // prints 3
cout << &myArray[0]; // prints 3

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;
```

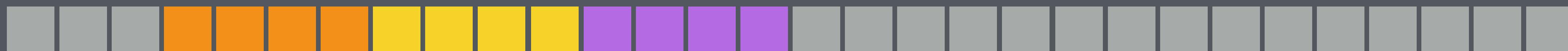


```
float *myArray = new float[3];
```

myArray



myArray[0]    myArray[1]    myArray[2]



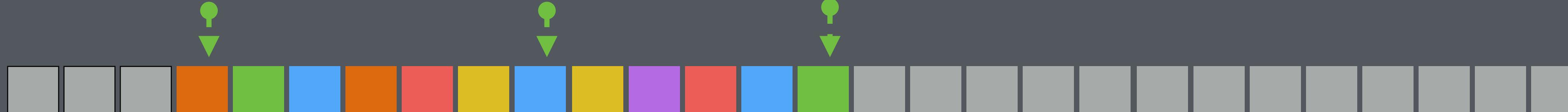
0    1    2    3    4    5    6    7    8    9    10    11    12    13    14    15    16

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

myCharArray



myCharArray[0]    myCharArray[6]    myCharArray[11]



0    1    2    3    4    5    6    7    8    9    10    11    12    13    14    15    16

Image on the screen

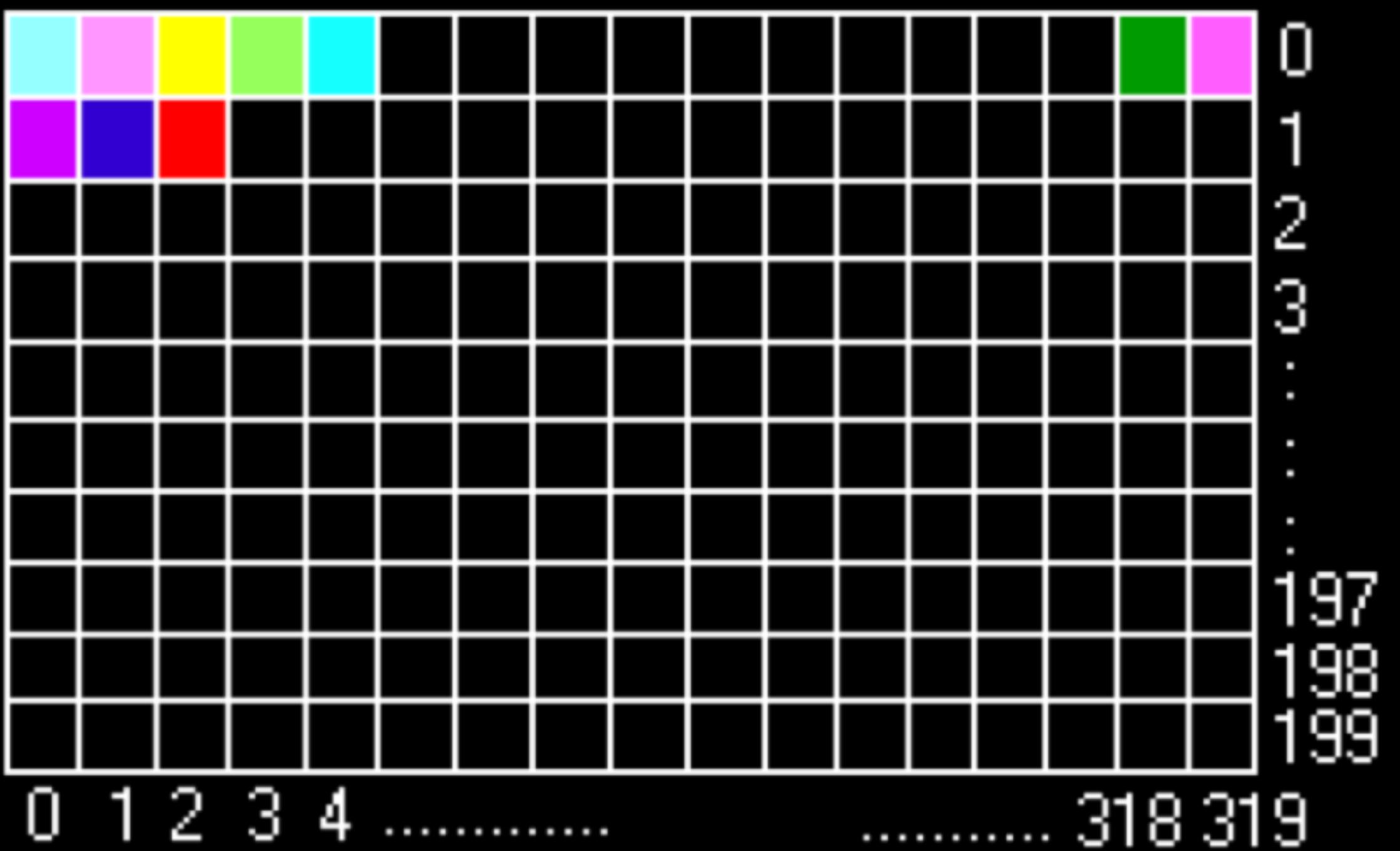


Image in memory



# Graphics in games

Defend

Put Away

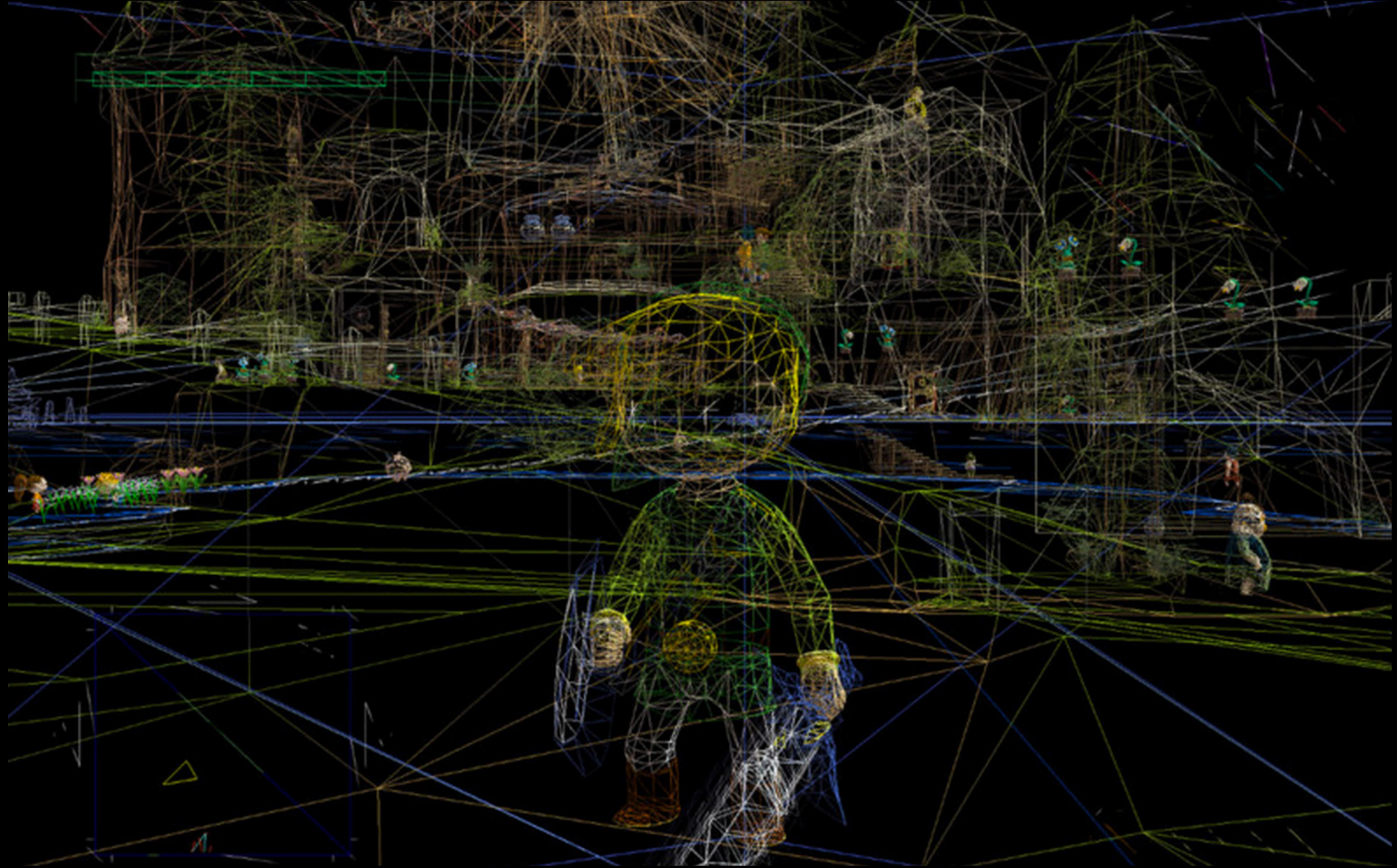
76

72

FREE

4168



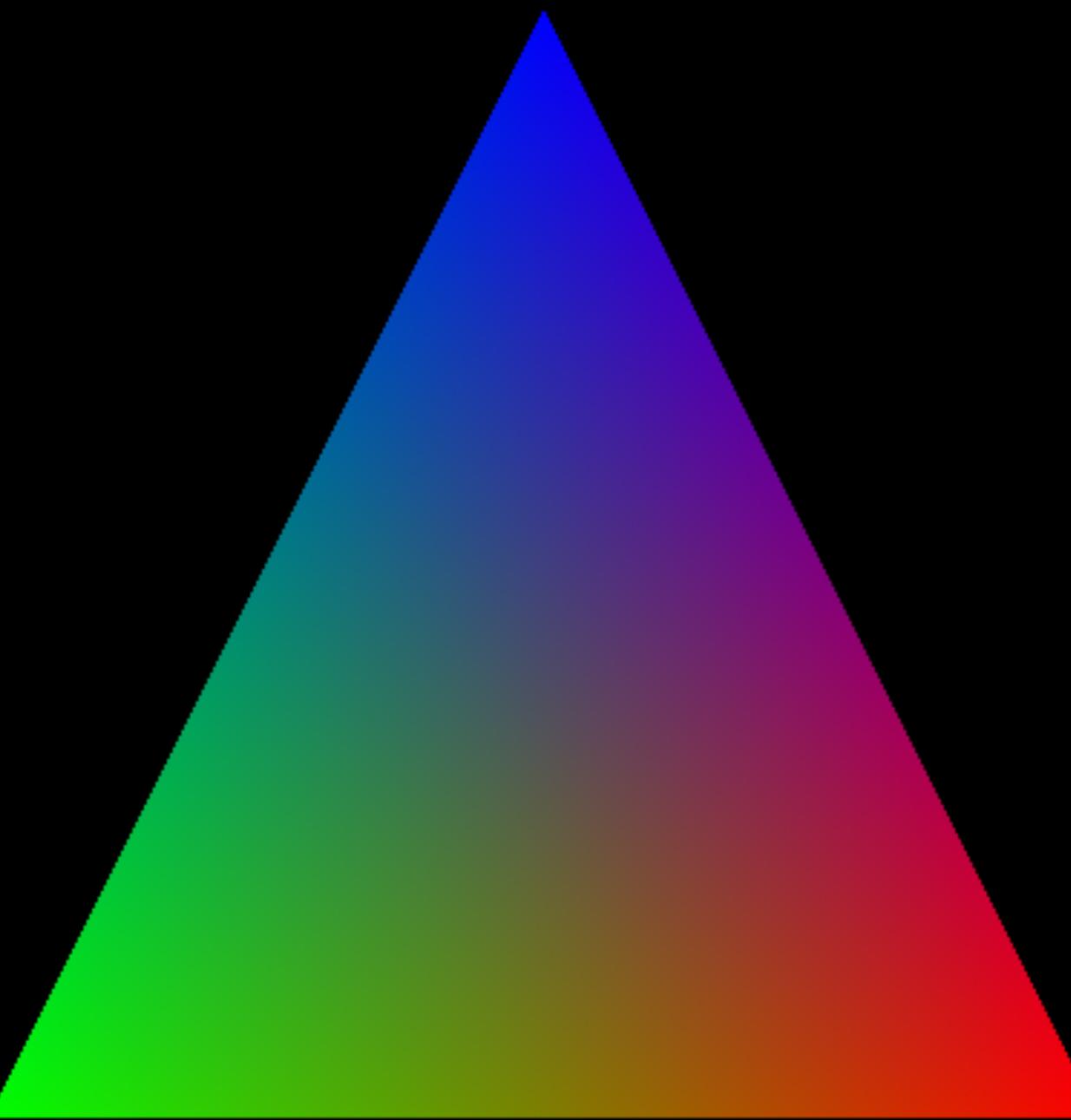








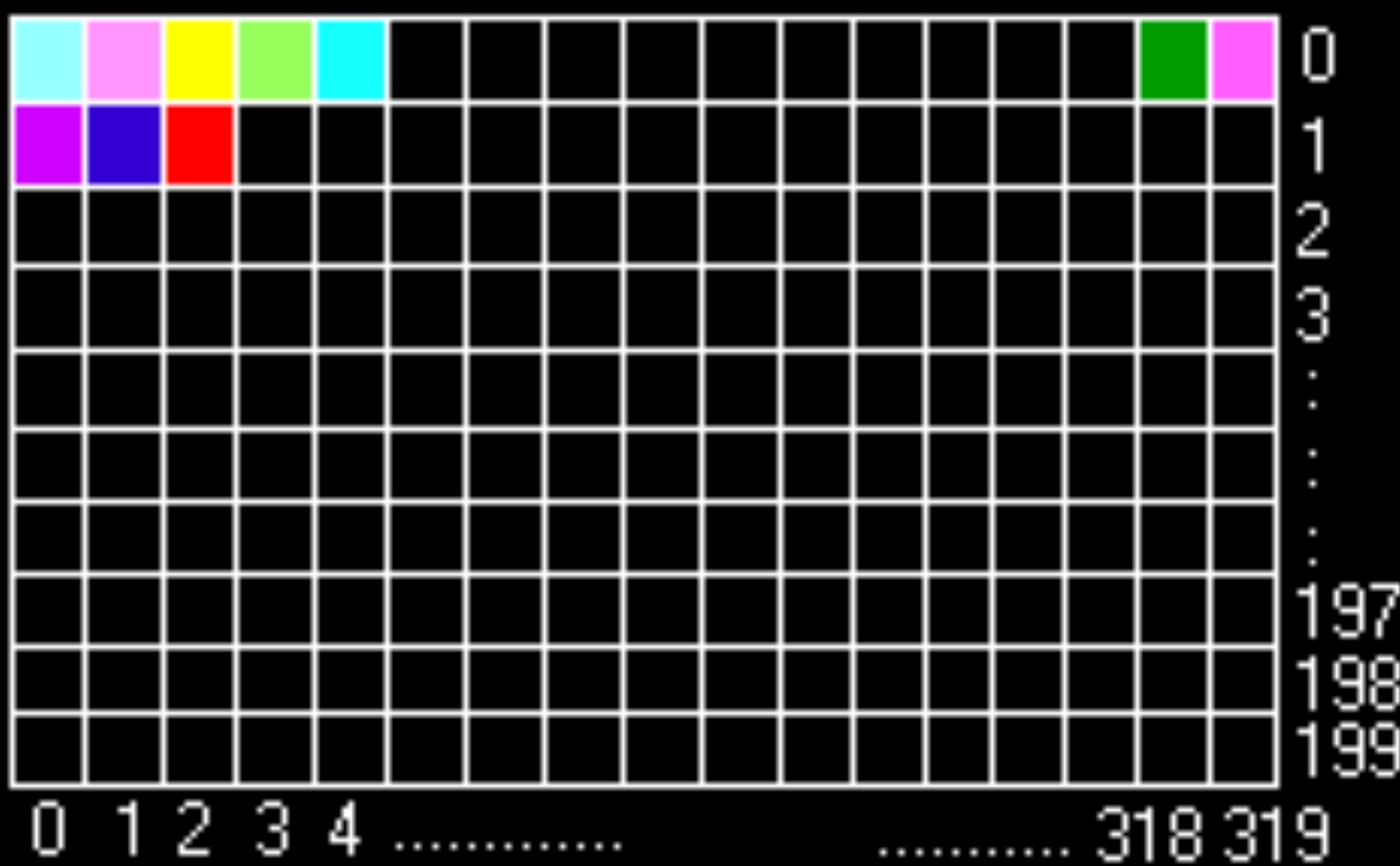




# Why triangles?



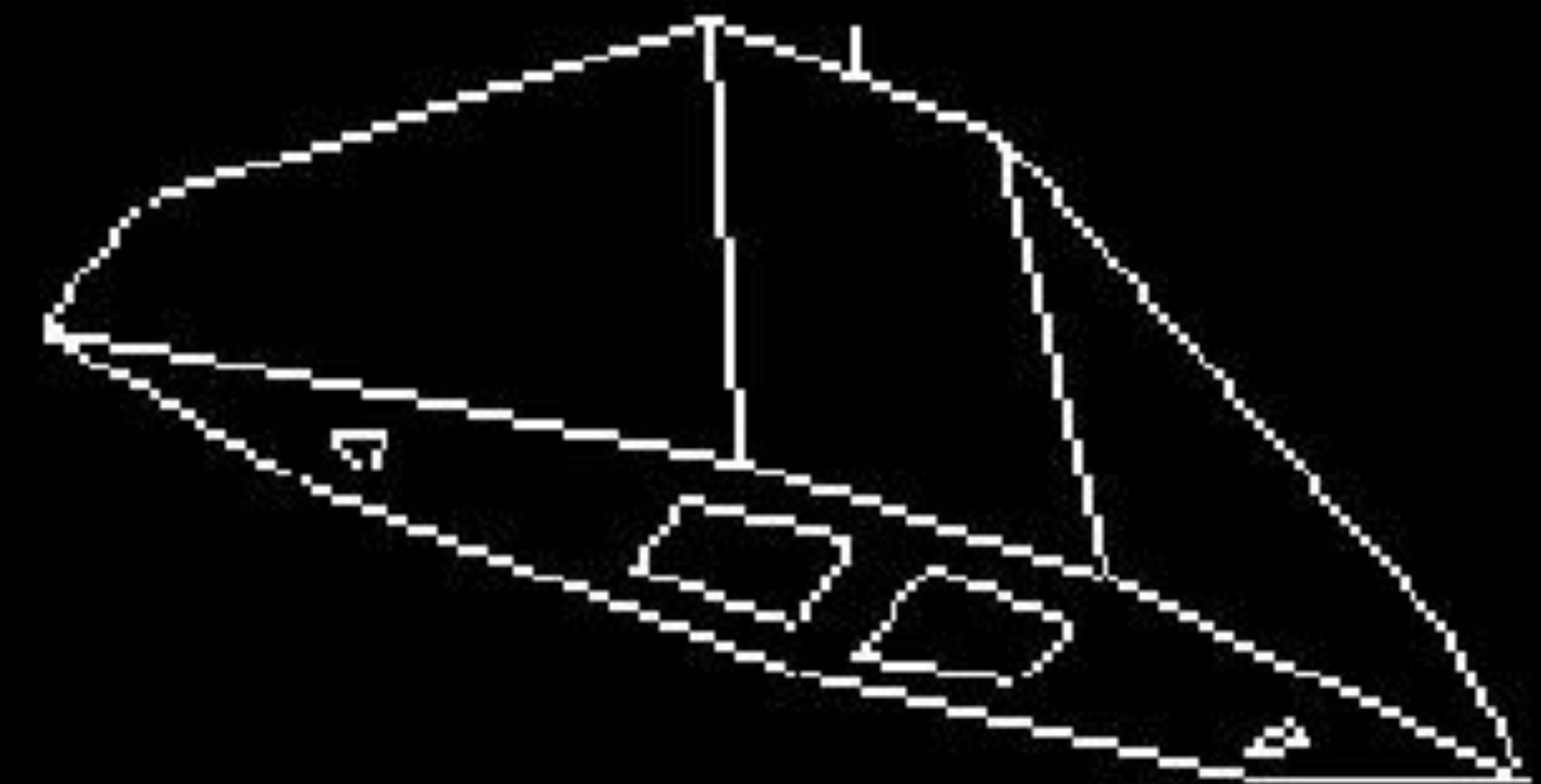
The screen



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



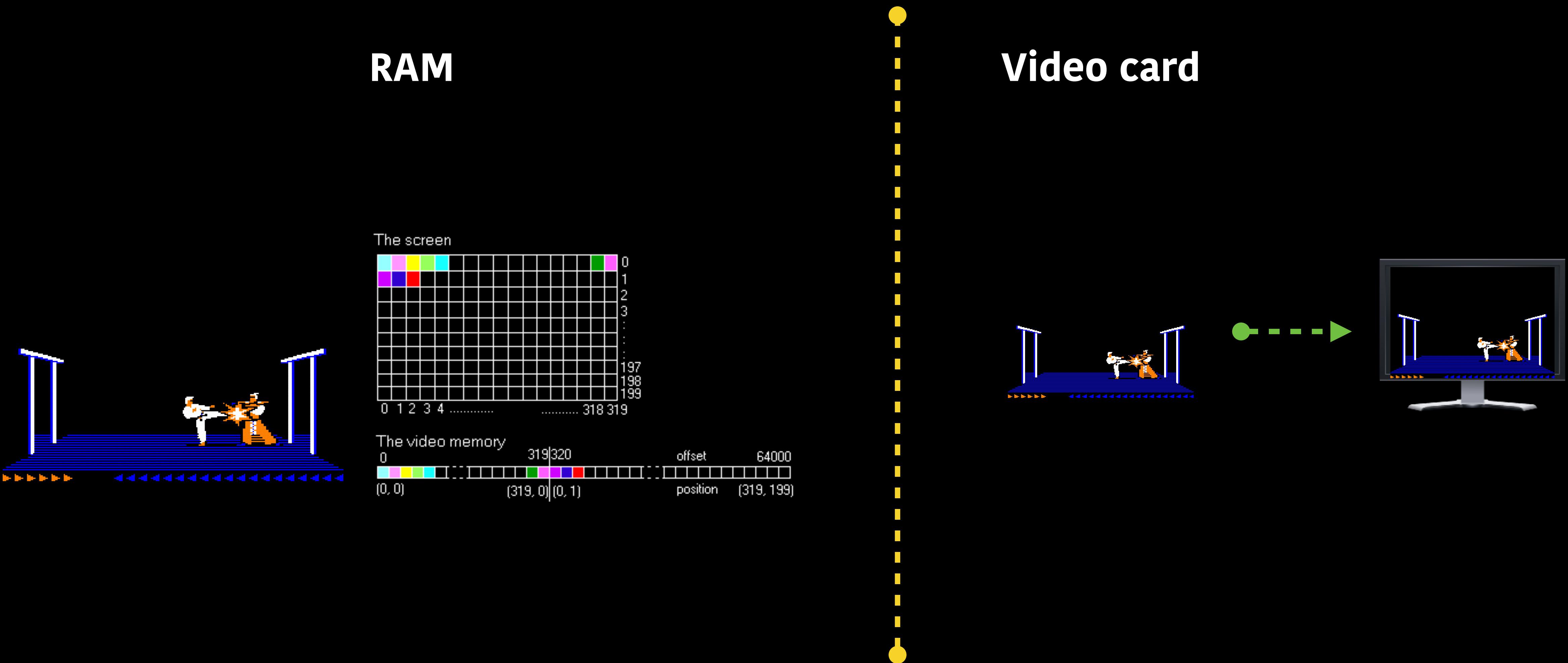
RED STORM  
TOTAL KILLS : 19  
RESISTS : LIGHTNING  
IMMUNE : MAGIC

INV

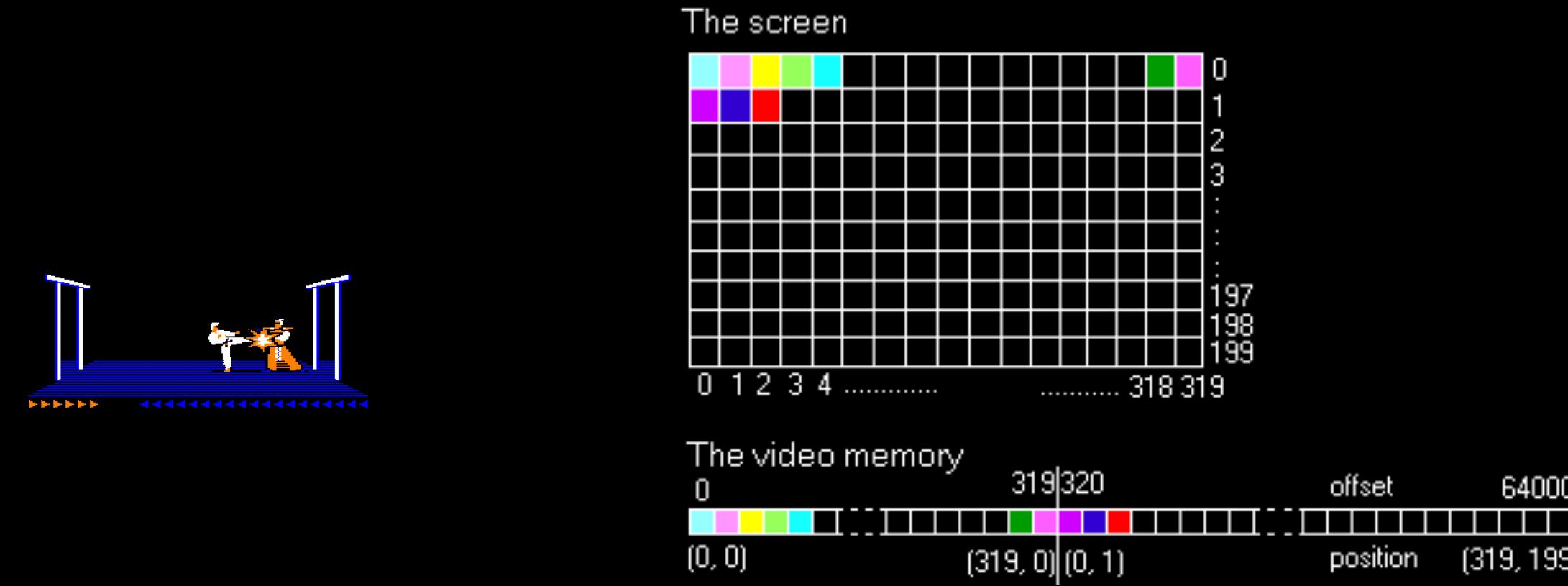
SPELLS



# Software Rendering



# Software Rendering



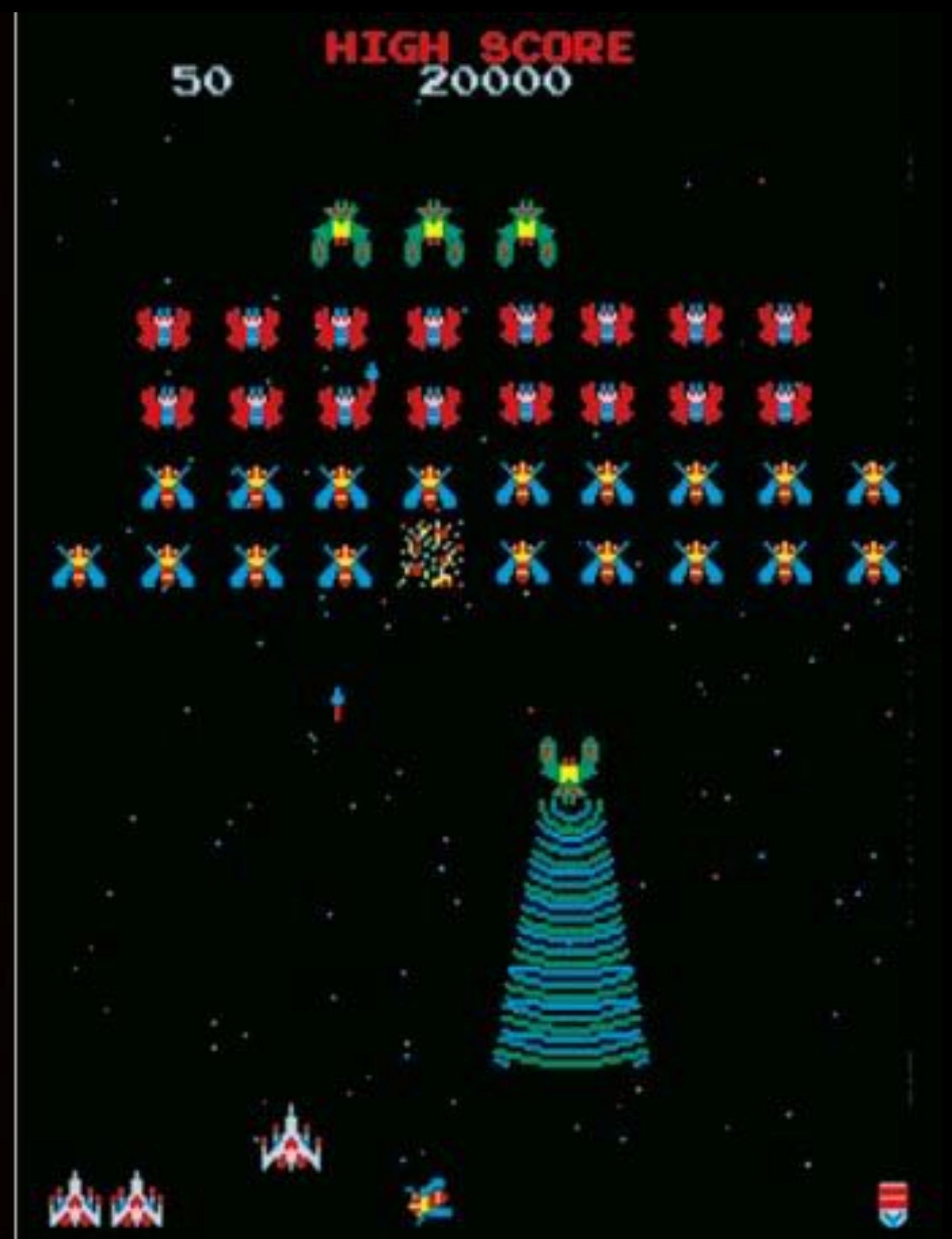
$2560 * 1600 = 4,096,000$  pixels

$\sim 12$  MB ==  $4 * \text{sizeof}(\text{War and Peace})$

60 times a second

# Hardware rendering







MARIO  
000000

0 x 00

WORLD TIME  
1-1

# SUPER MARIO BROS.

@1985 NINTENDO

• 1 PLAYER GAME

2 PLAYER GAME

TOP - 000000

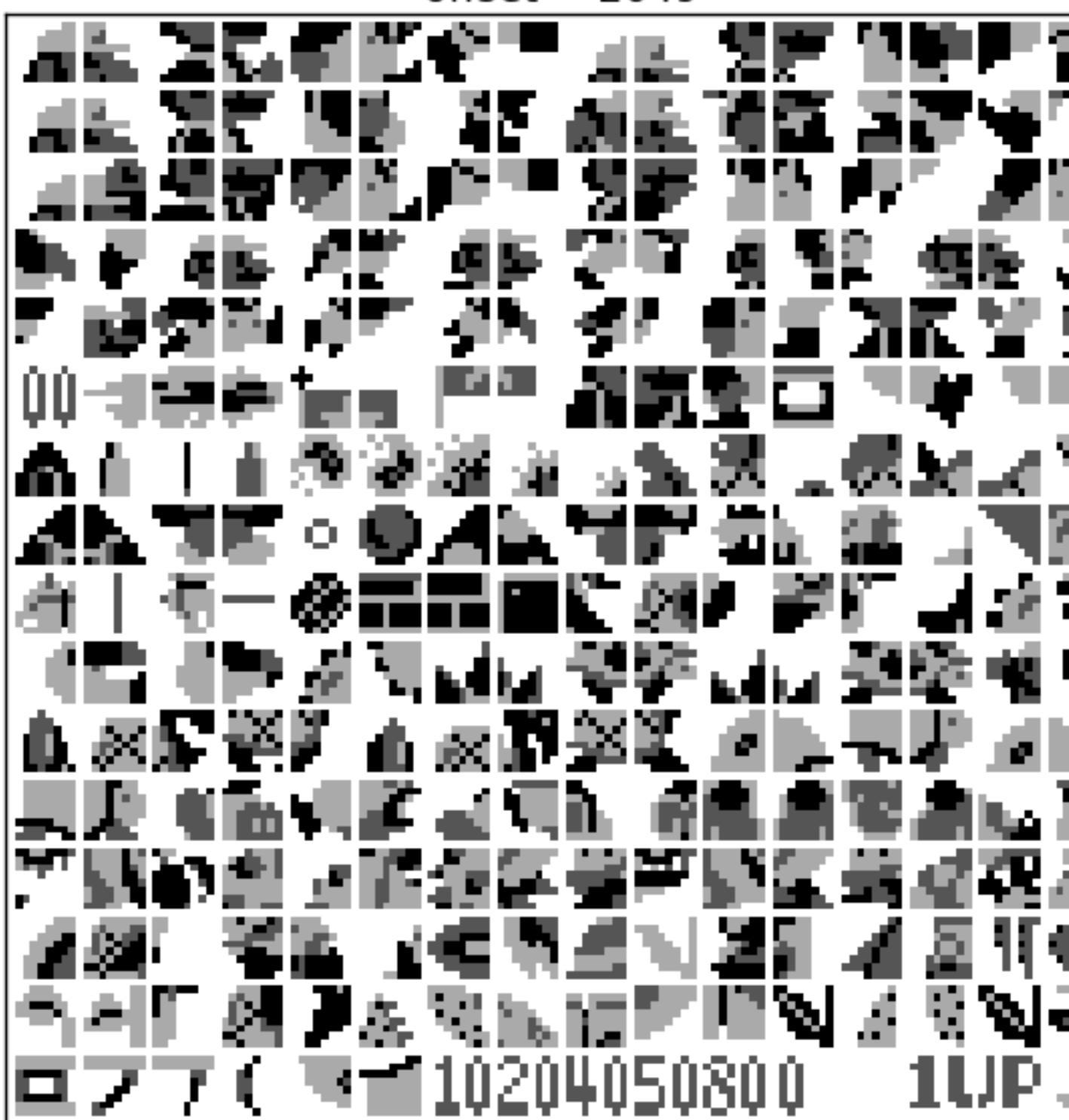


offset = 2305



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



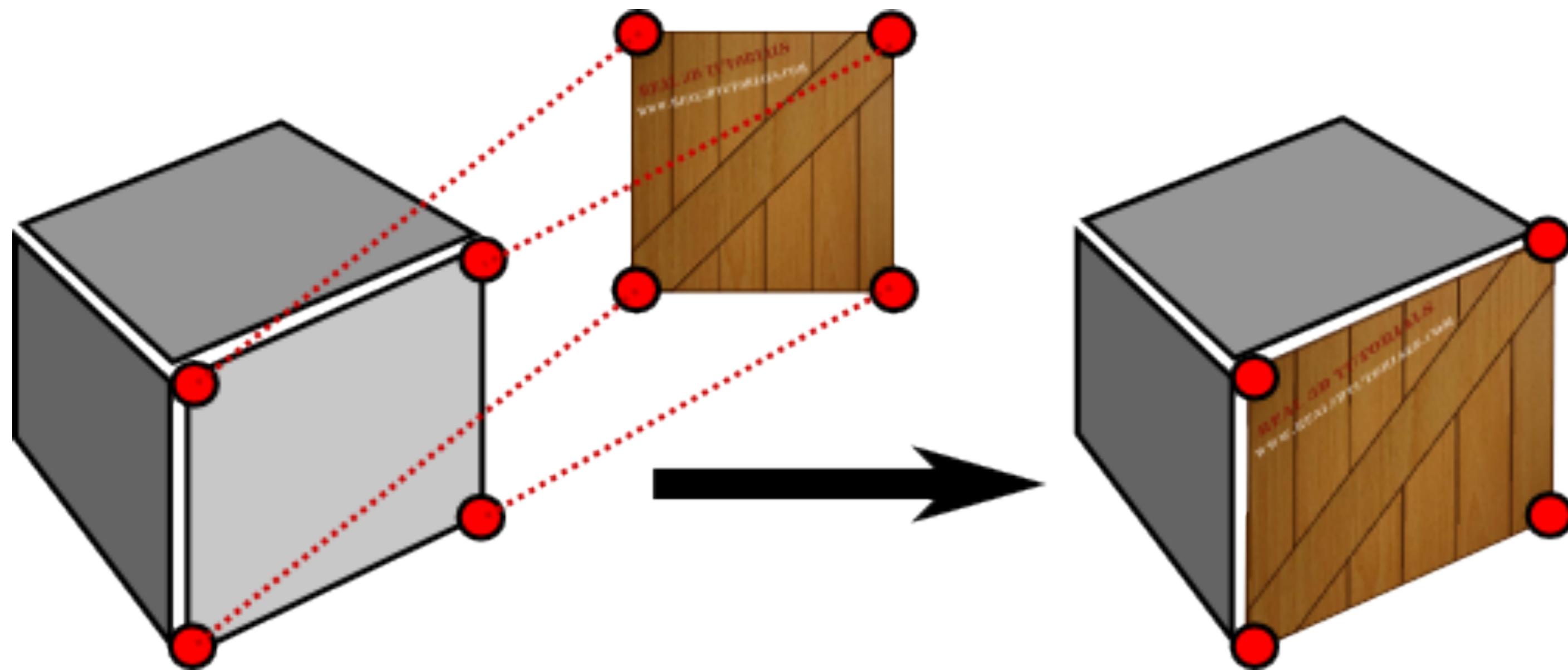
\$23C0	\$23C1	\$23C2	\$23C3	\$23C4	\$23C5	\$23C6	\$23C7
000000	000000	000000	000000	000000	000000	000000	000000
\$23C8	\$23C9	\$23CA	\$23CB	\$23CC	\$23CD	\$23CE	\$23CF
\$23D0	\$23D1	\$23D2	\$23D3	\$23D4	\$23D5	\$23D6	\$23D7
\$23D8	\$23D9	\$23DA	\$23DB	\$23DC	\$23DD	\$23DE	\$23DF
\$23E0	\$23E1	\$23E2	\$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**

**Get the Magic of Speed!**

**4**

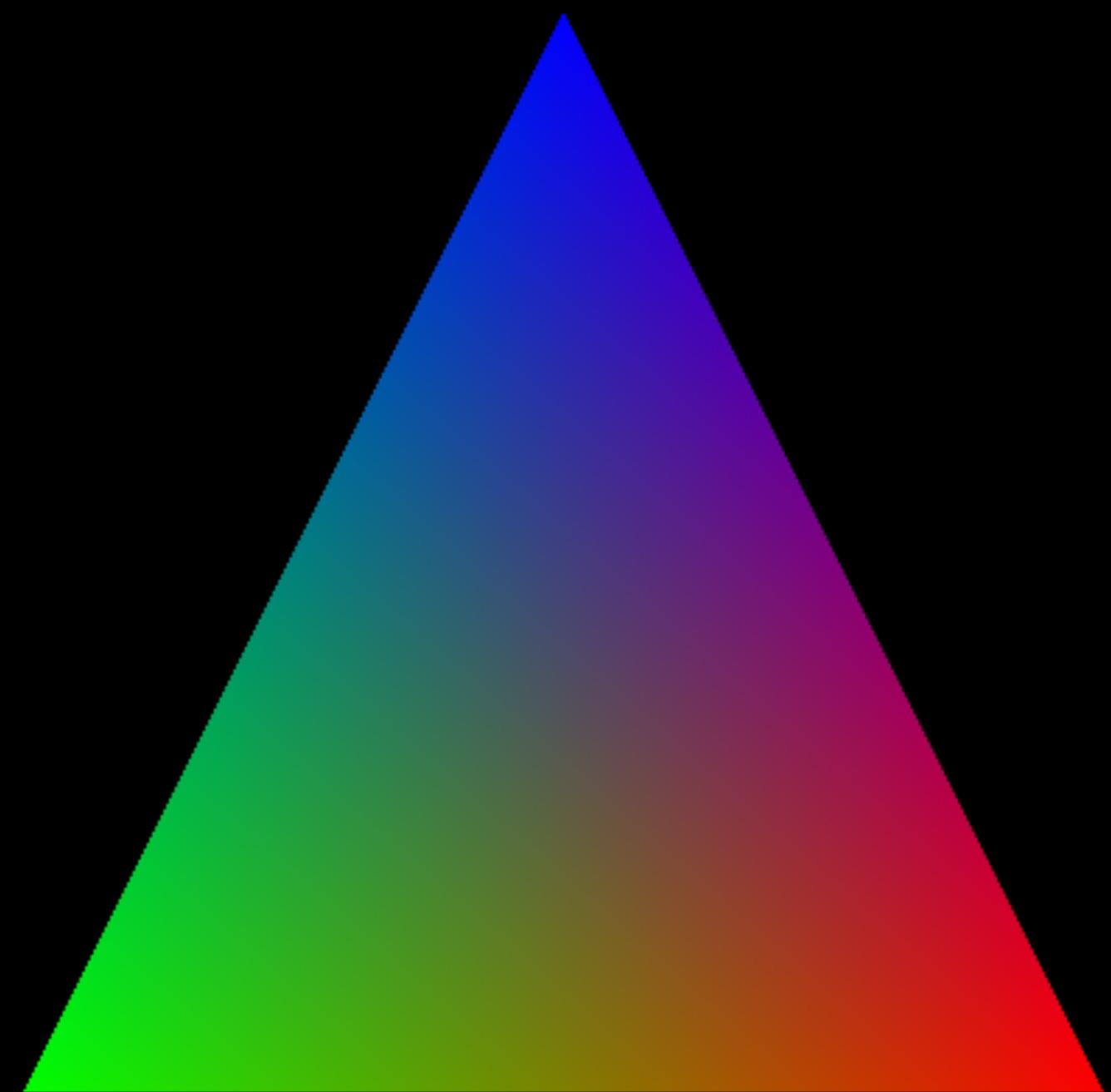
**CREATIVE**  
www.soundblaster.com

- 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







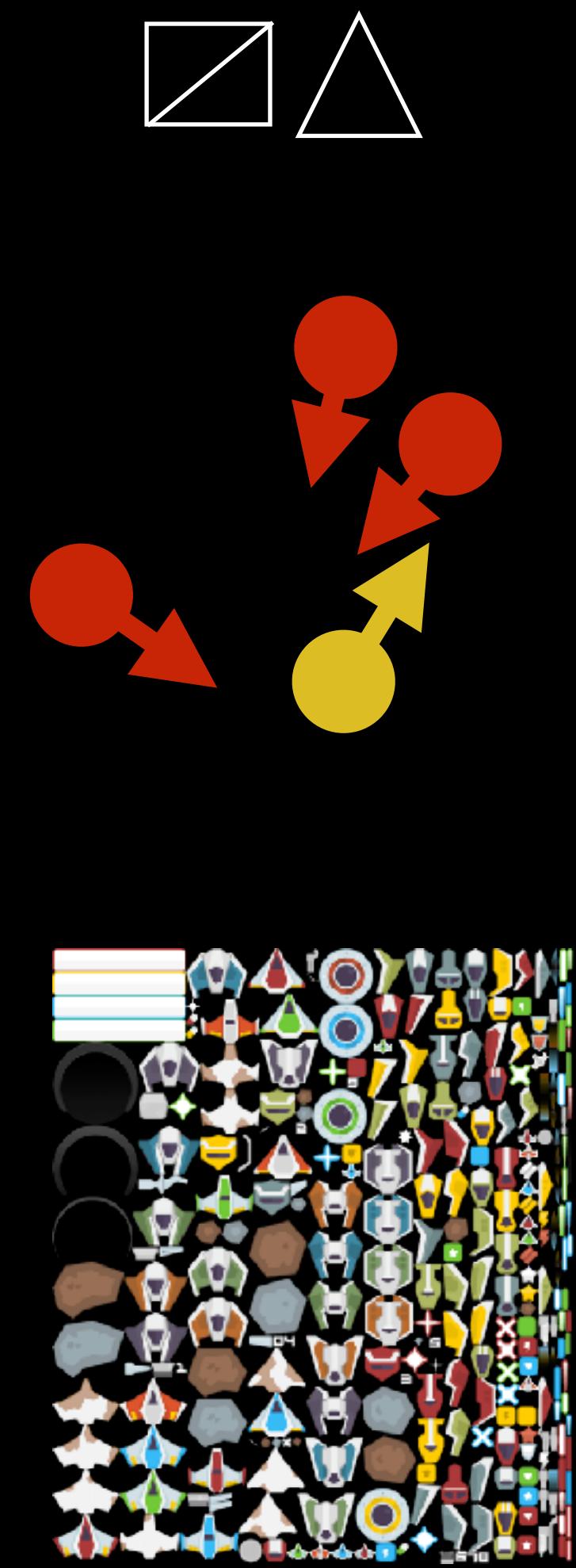


# Hardware Rendering



# Hardware Rendering

RAM



Video card

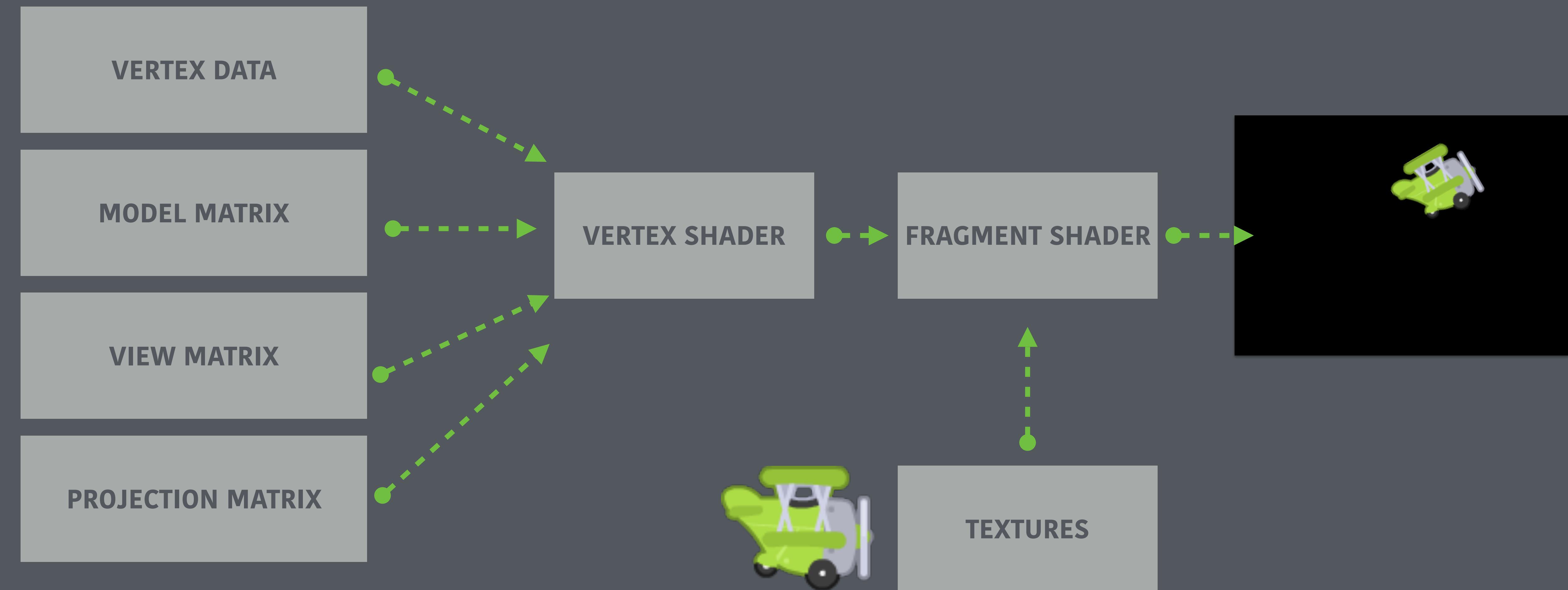
# The GPU pipeline



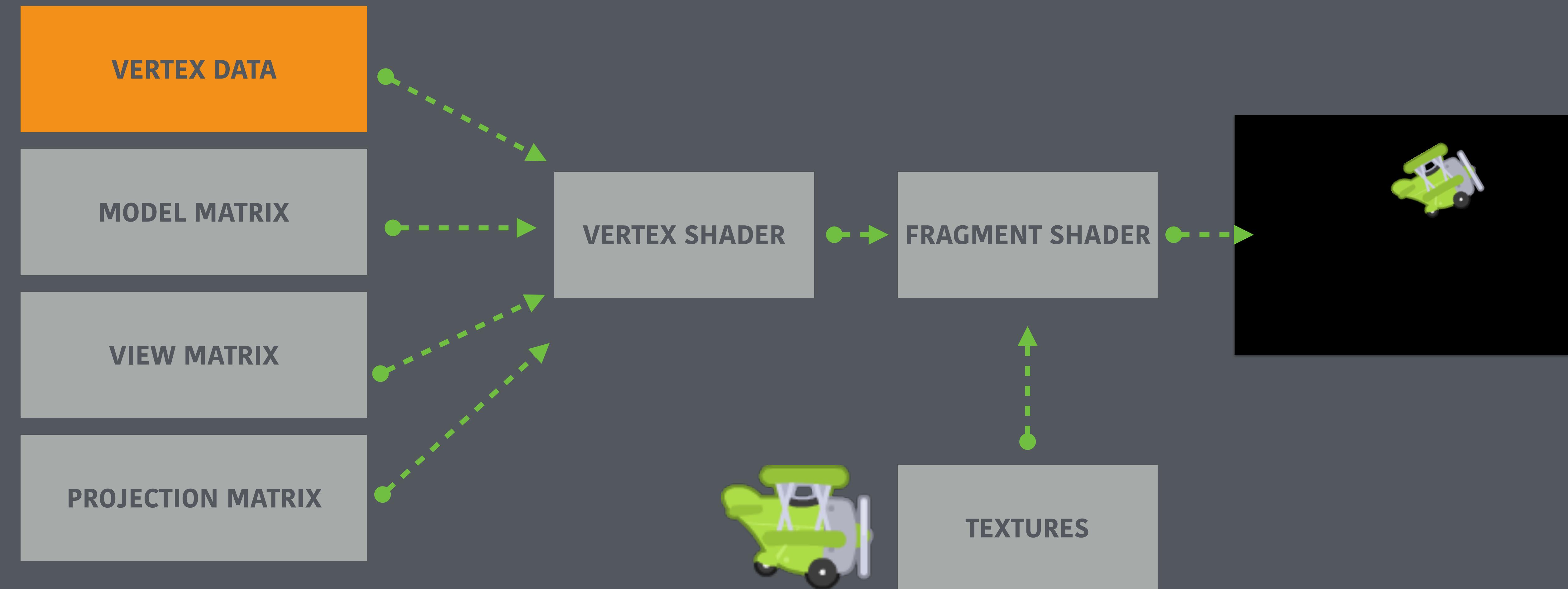


What?  
Where?  
How?

# The GPU pipeline

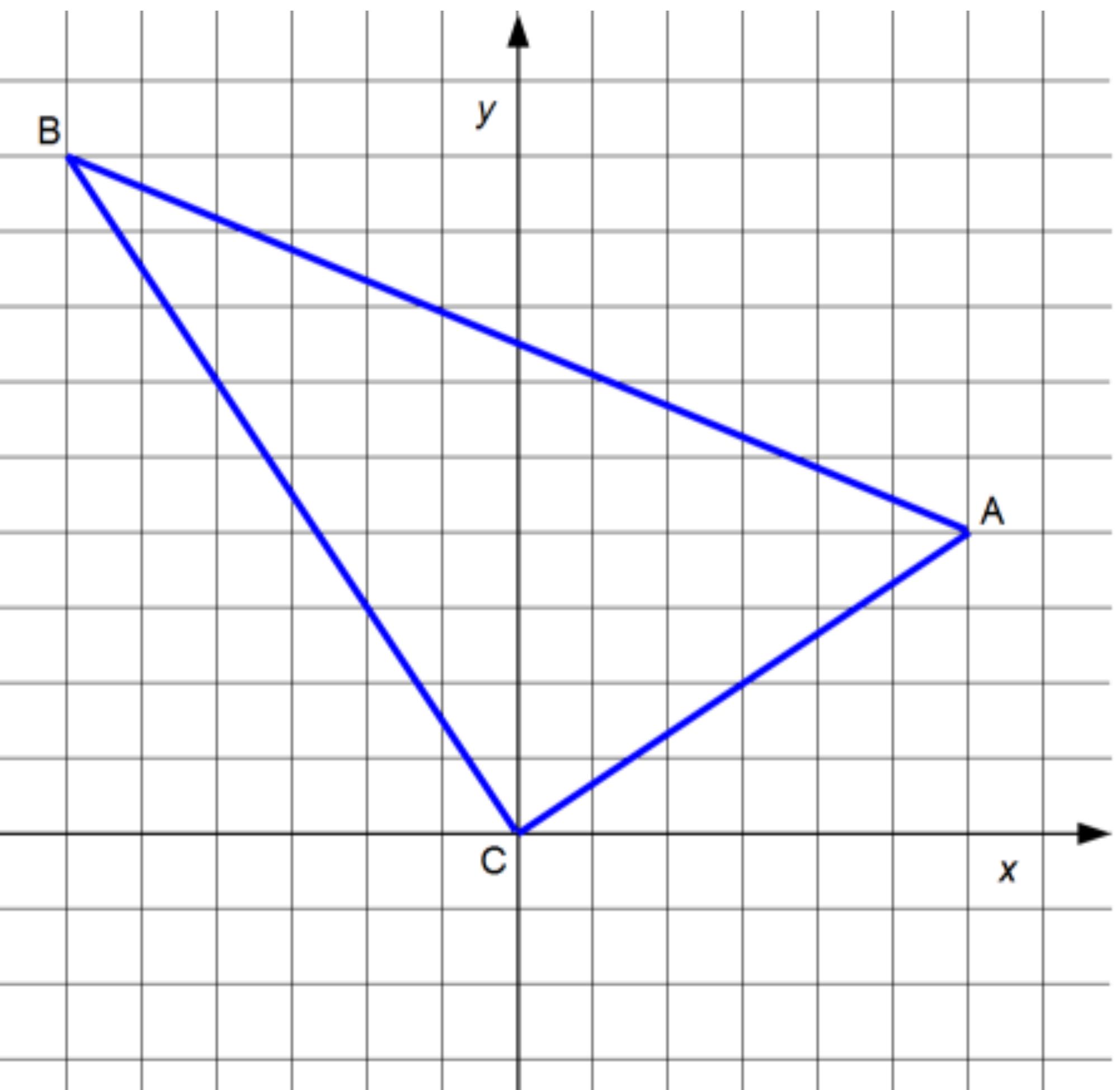


# The GPU pipeline

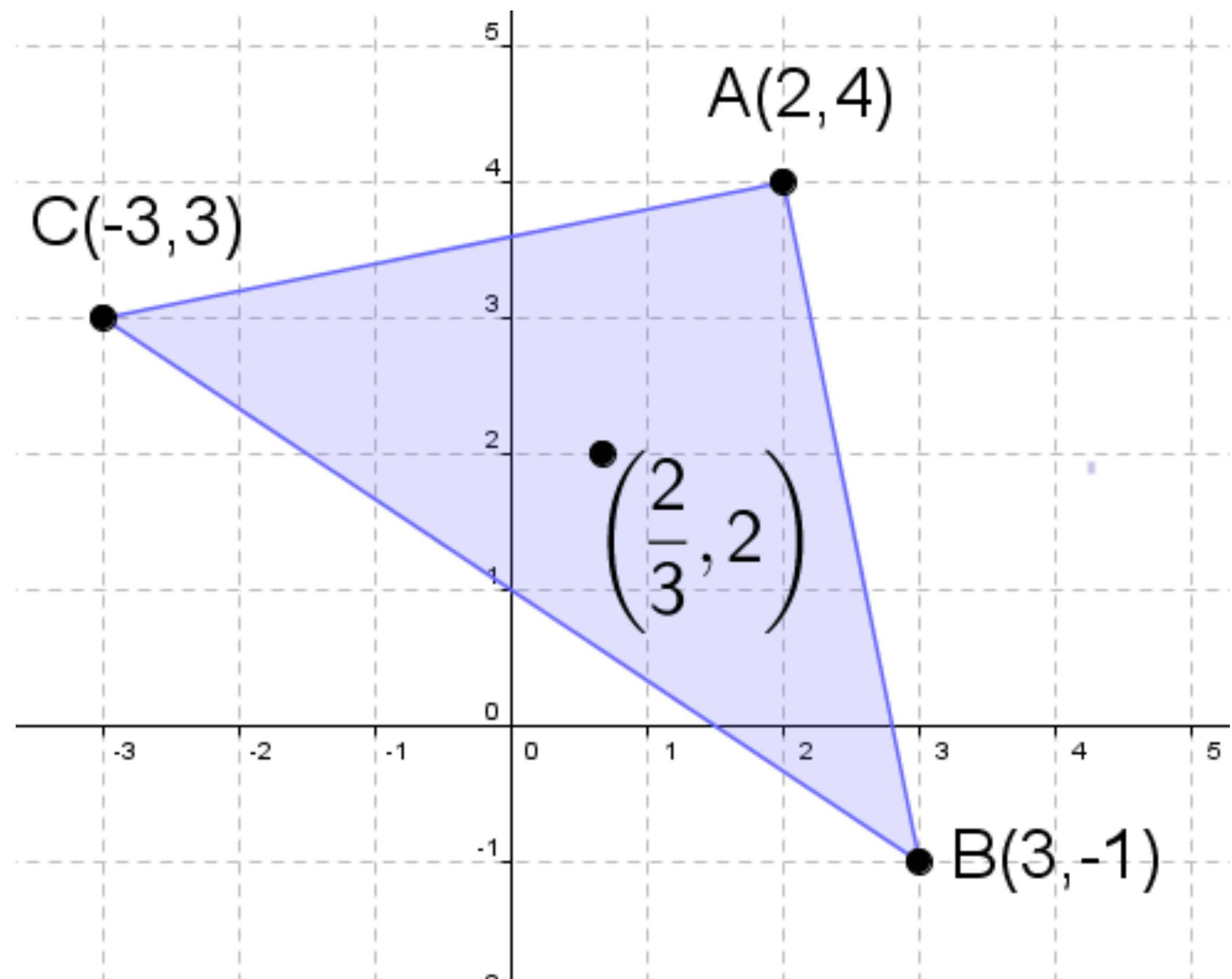


# Vertex data





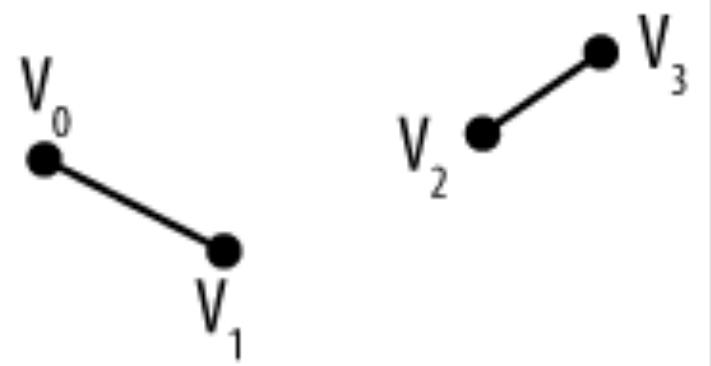
A polygon is defined by  
points in space called vertices



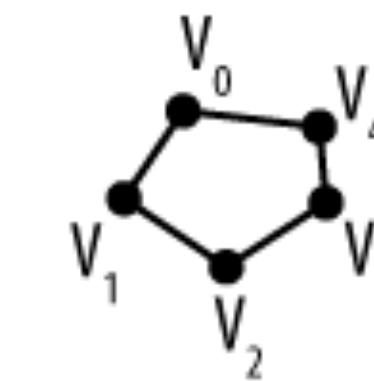
**GL\_POINTS**



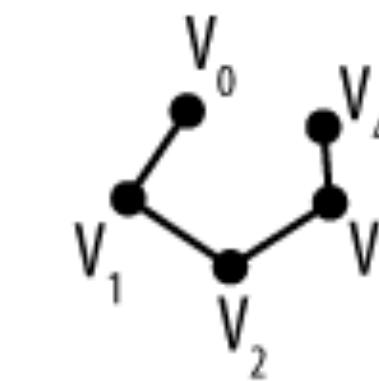
**GL\_LINES**



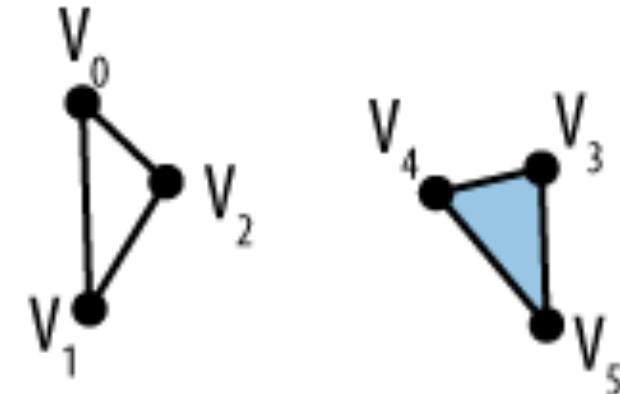
**GL\_LINE\_LOOP**



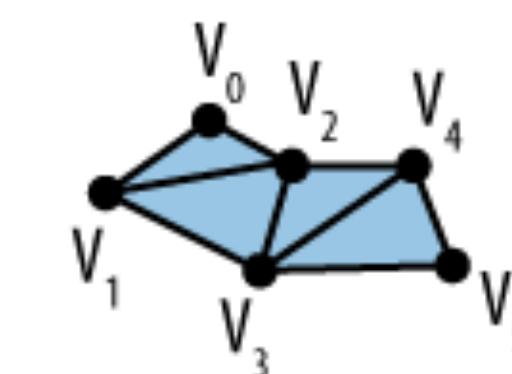
**GL\_LINE\_STRIP**



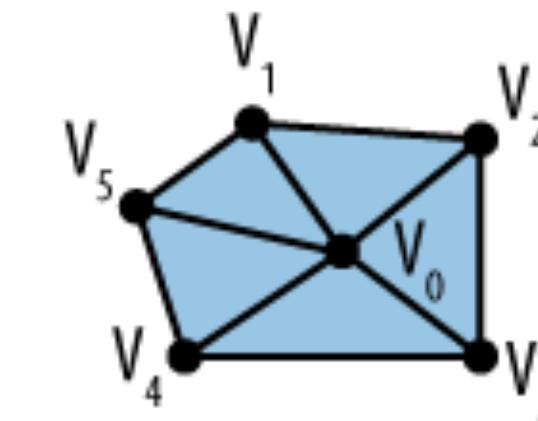
**GL\_TRIANGLES**



**GL\_TRIANGLE\_STRIP**



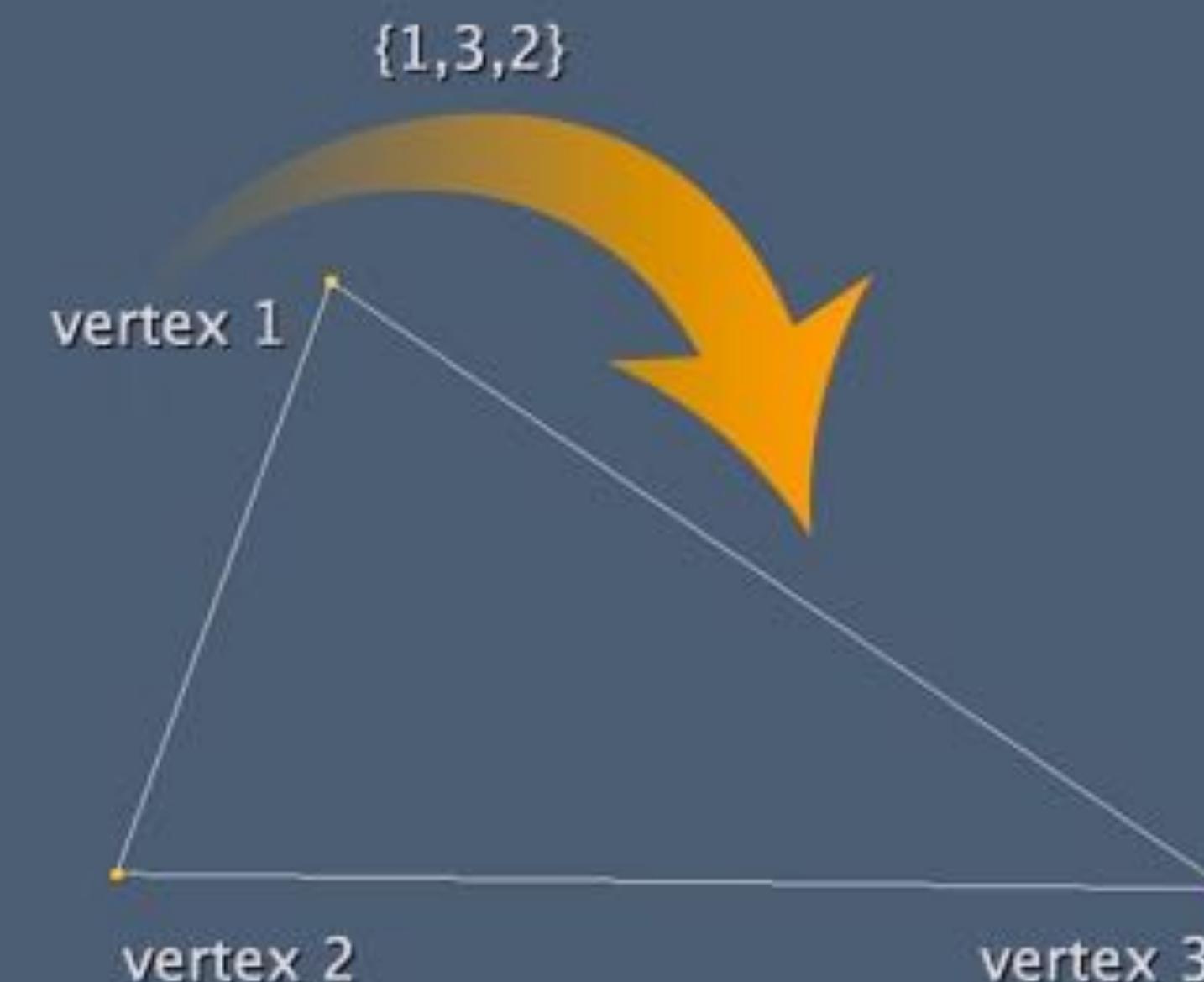
**GL\_TRIANGLE\_FAN**



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

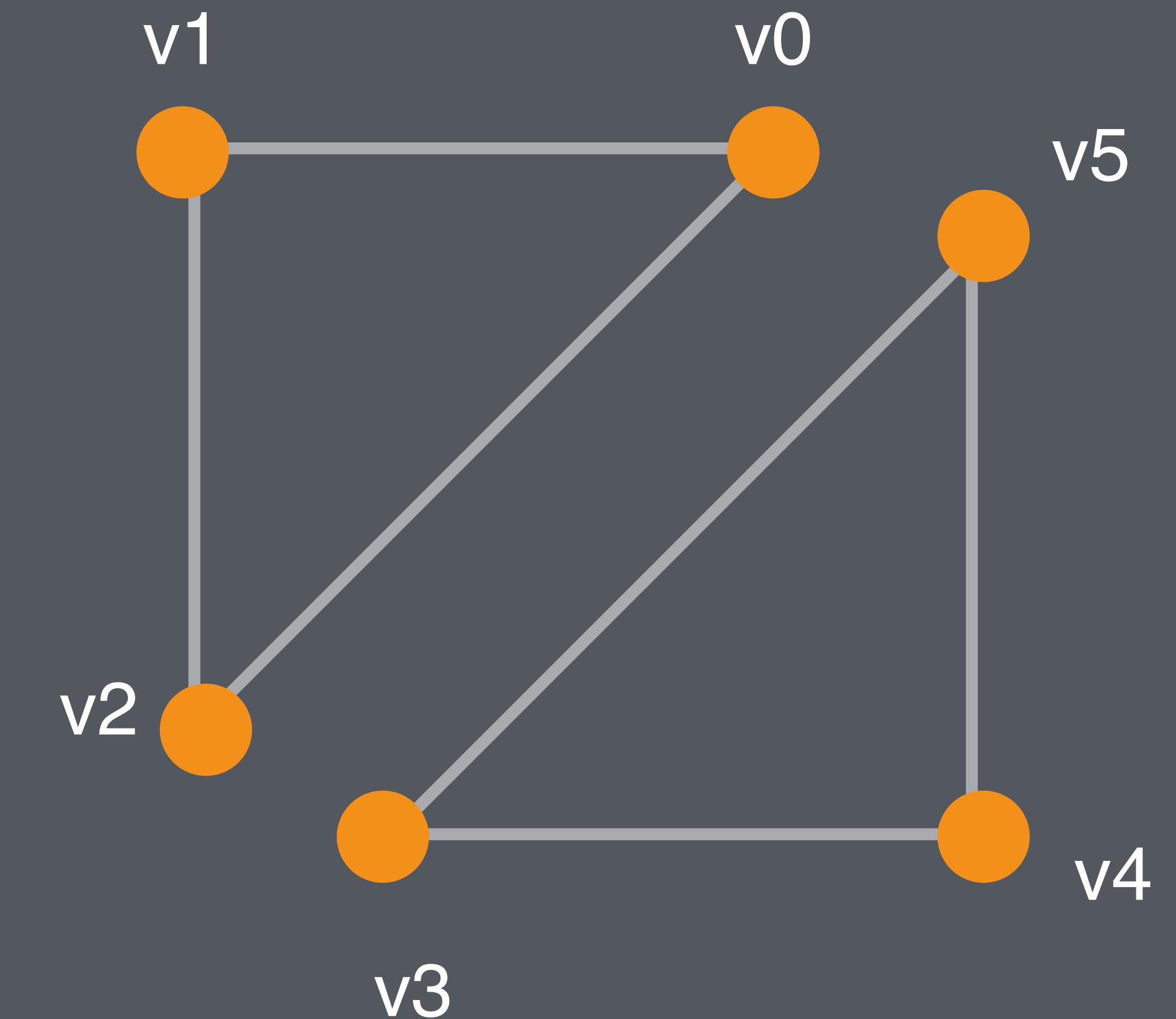


Front Facing Triangle  
(CCW)



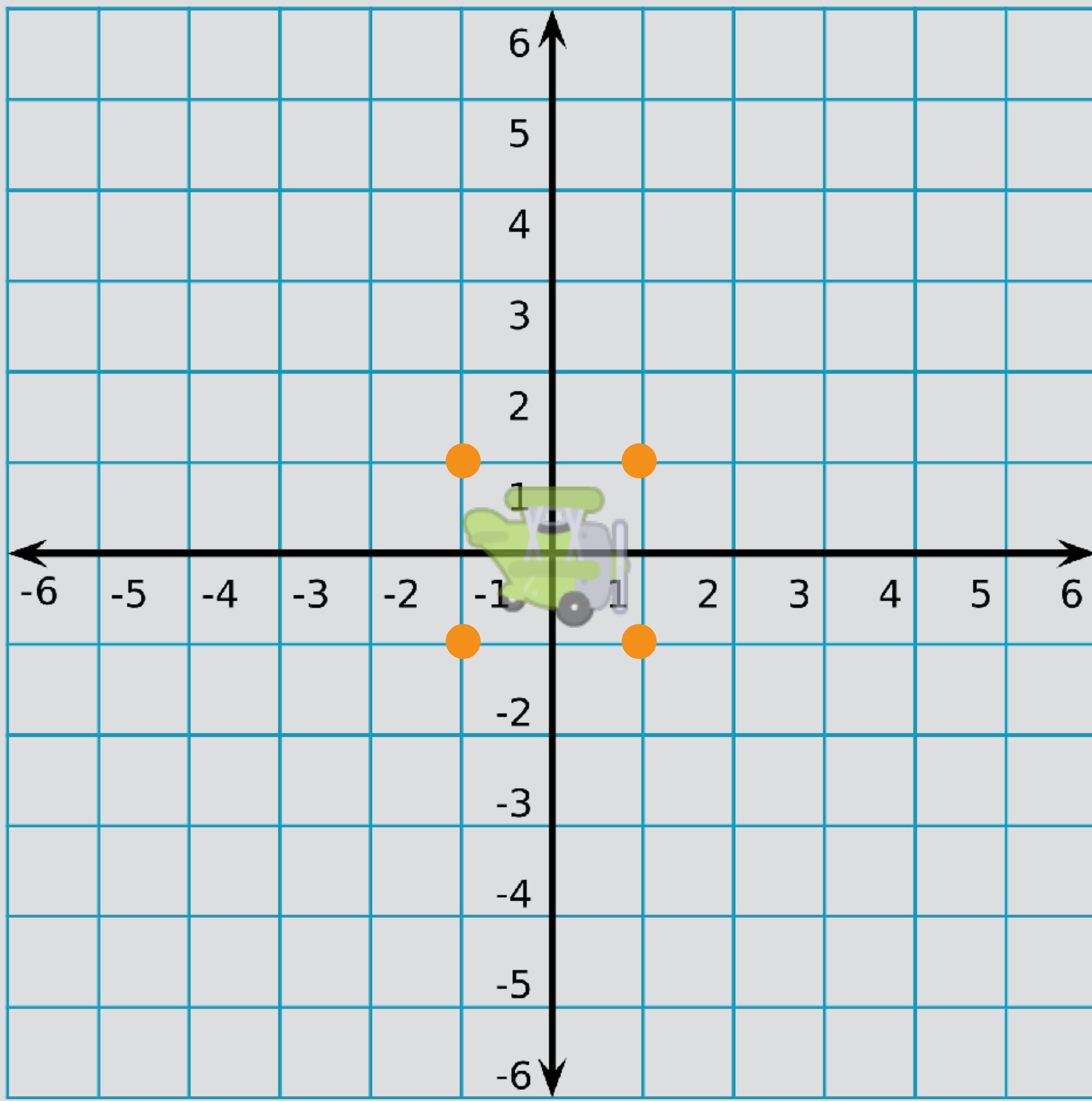
Back Facing Triangle  
(CW)

Counter-clockwise order!

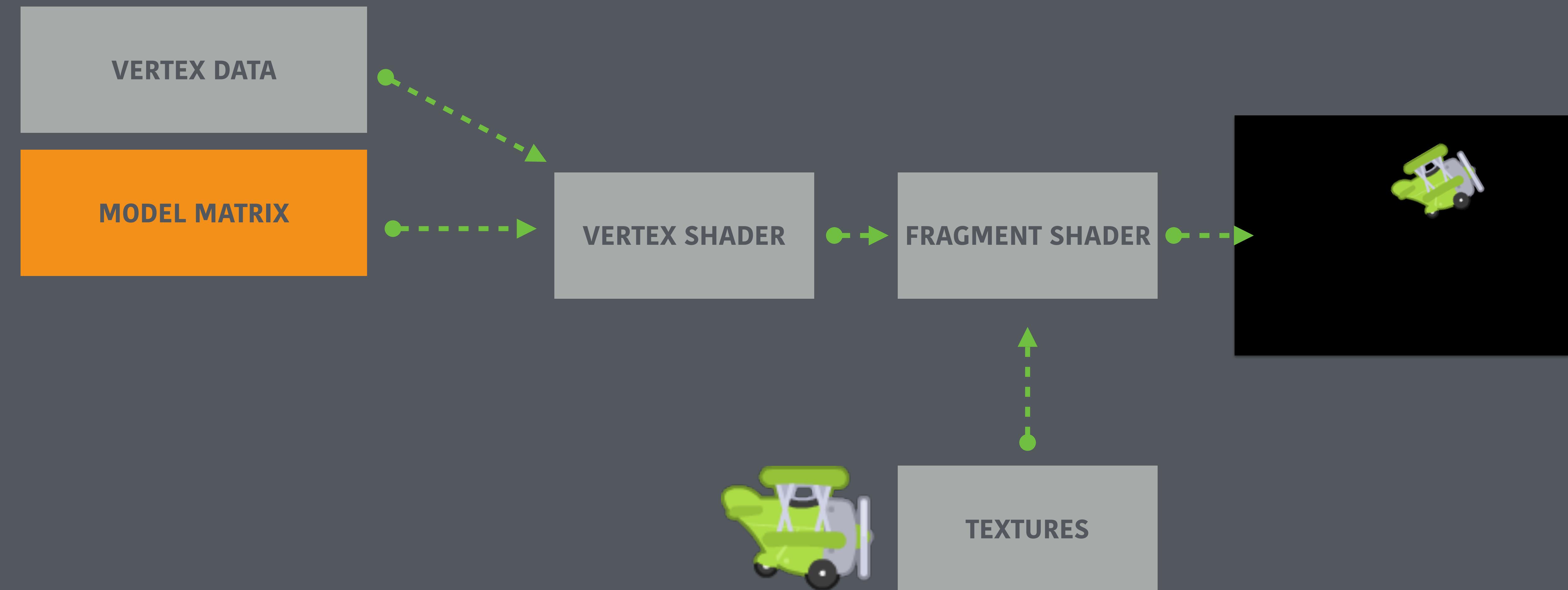


y-axis

x-axis



# The GPU pipeline



# The model matrix

# What is a matrix?

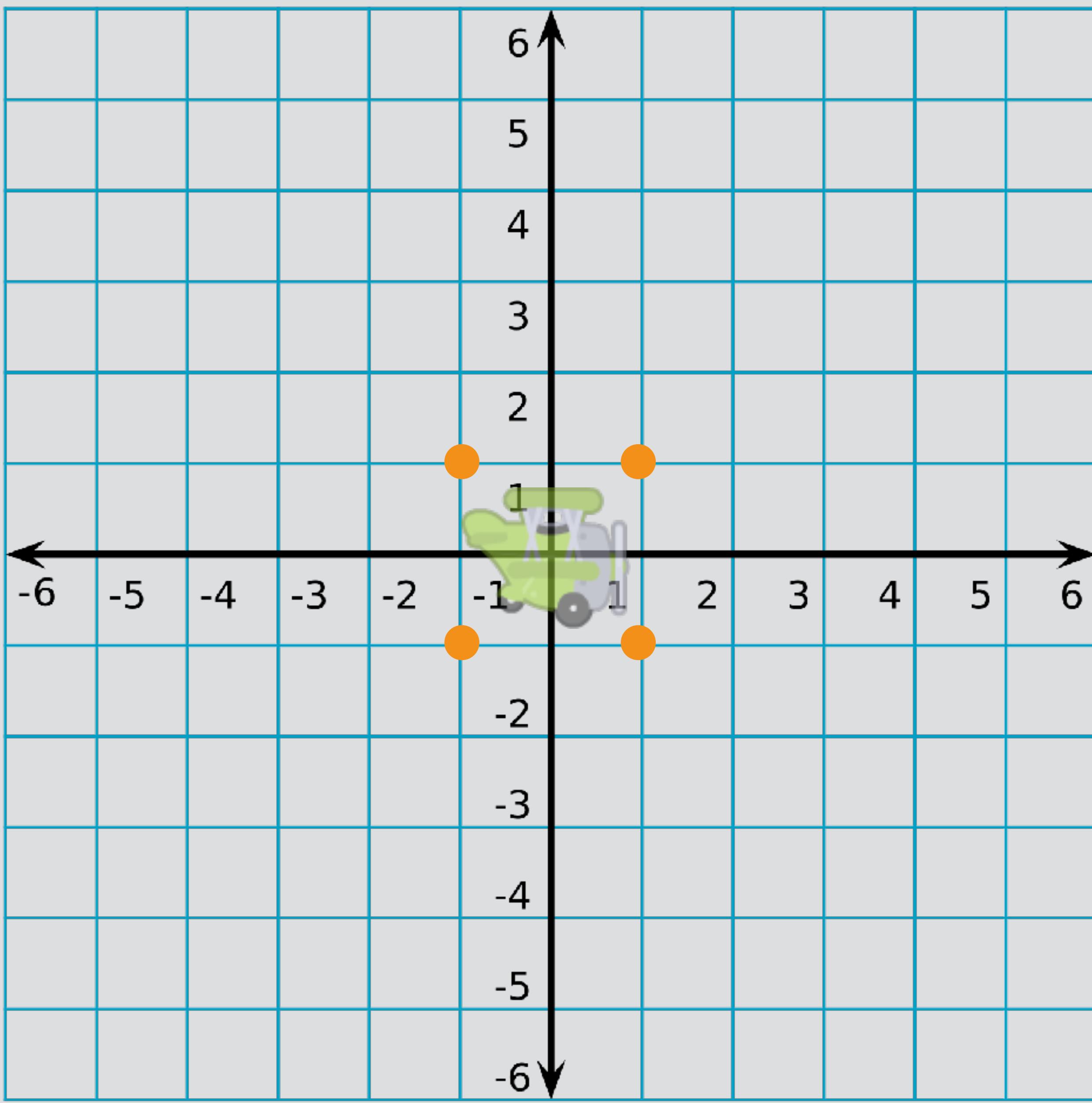
$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

# Transformation matrix

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

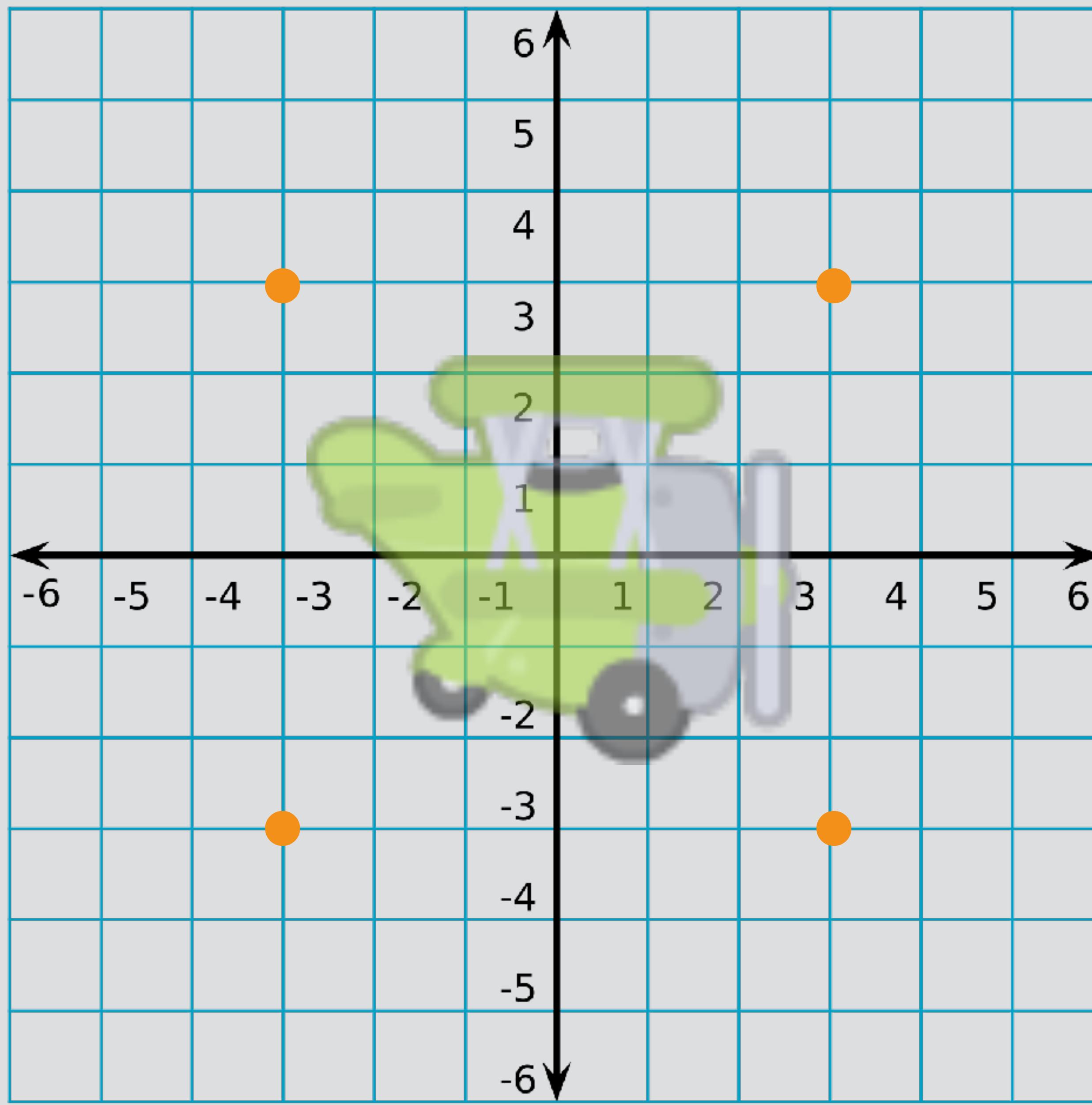
y-axis

x-axis



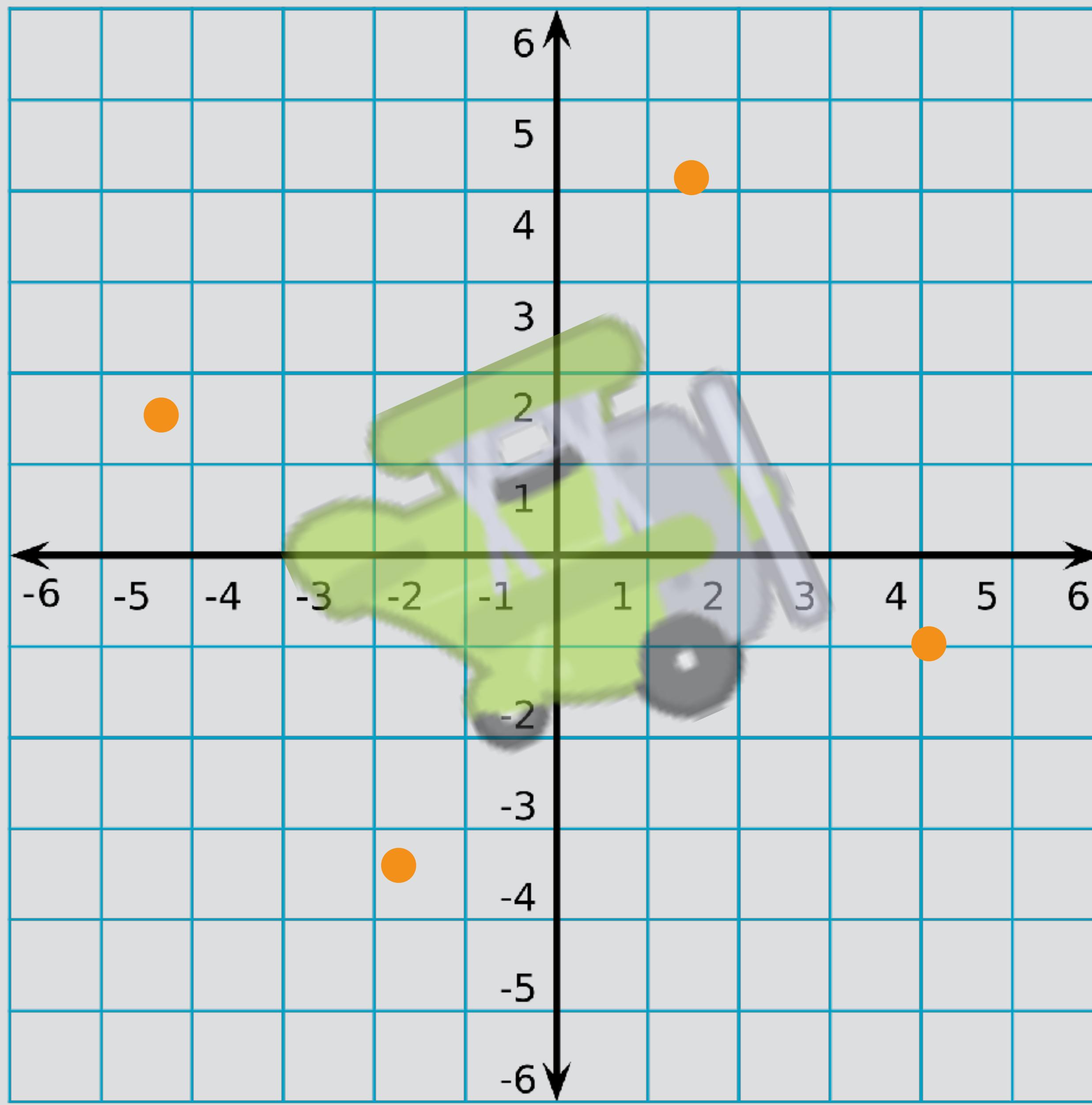
y-axis

x-axis



y-axis

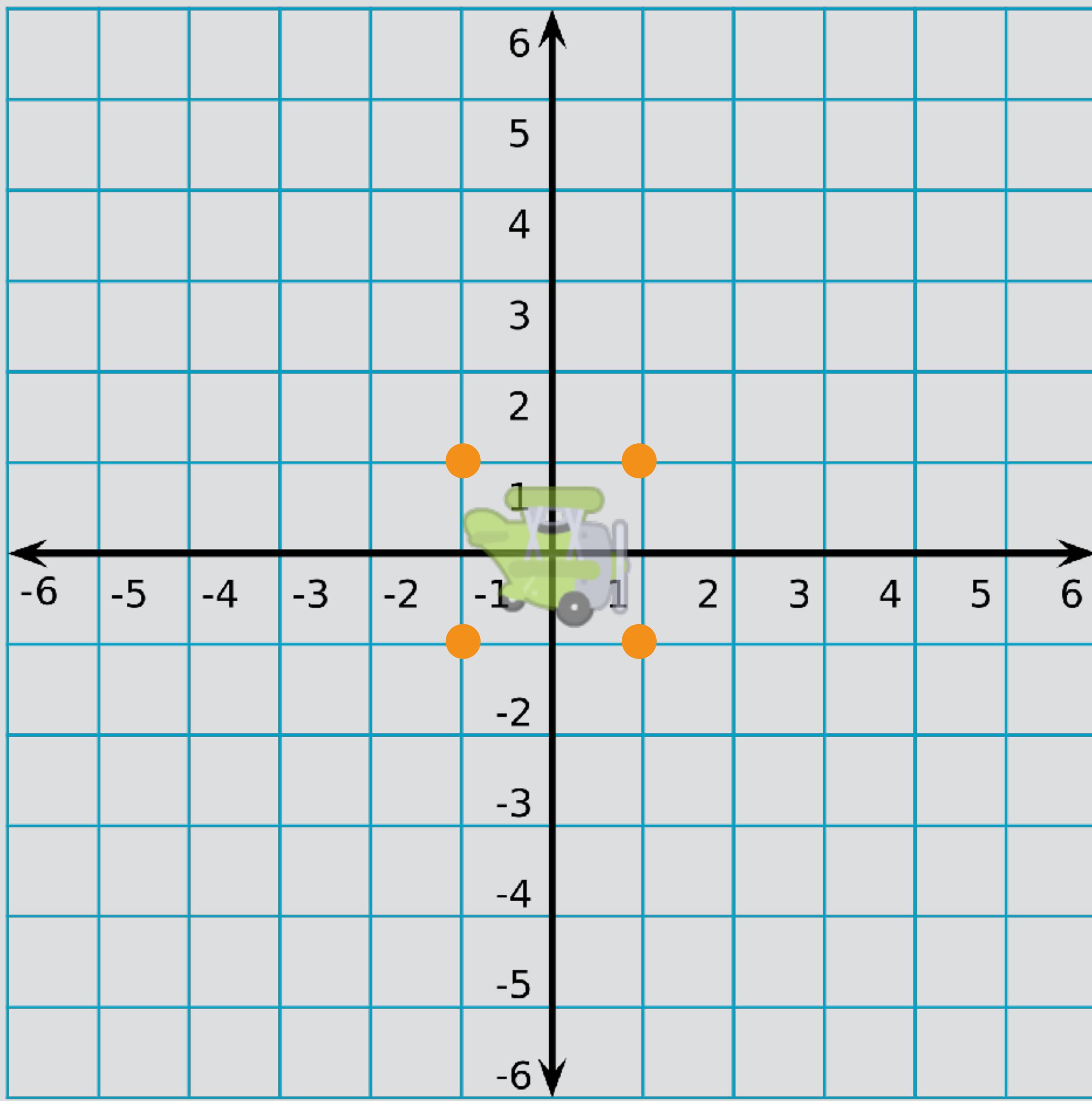
x-axis



An affine transformation matrix combines a linear transformation matrix with a translation.

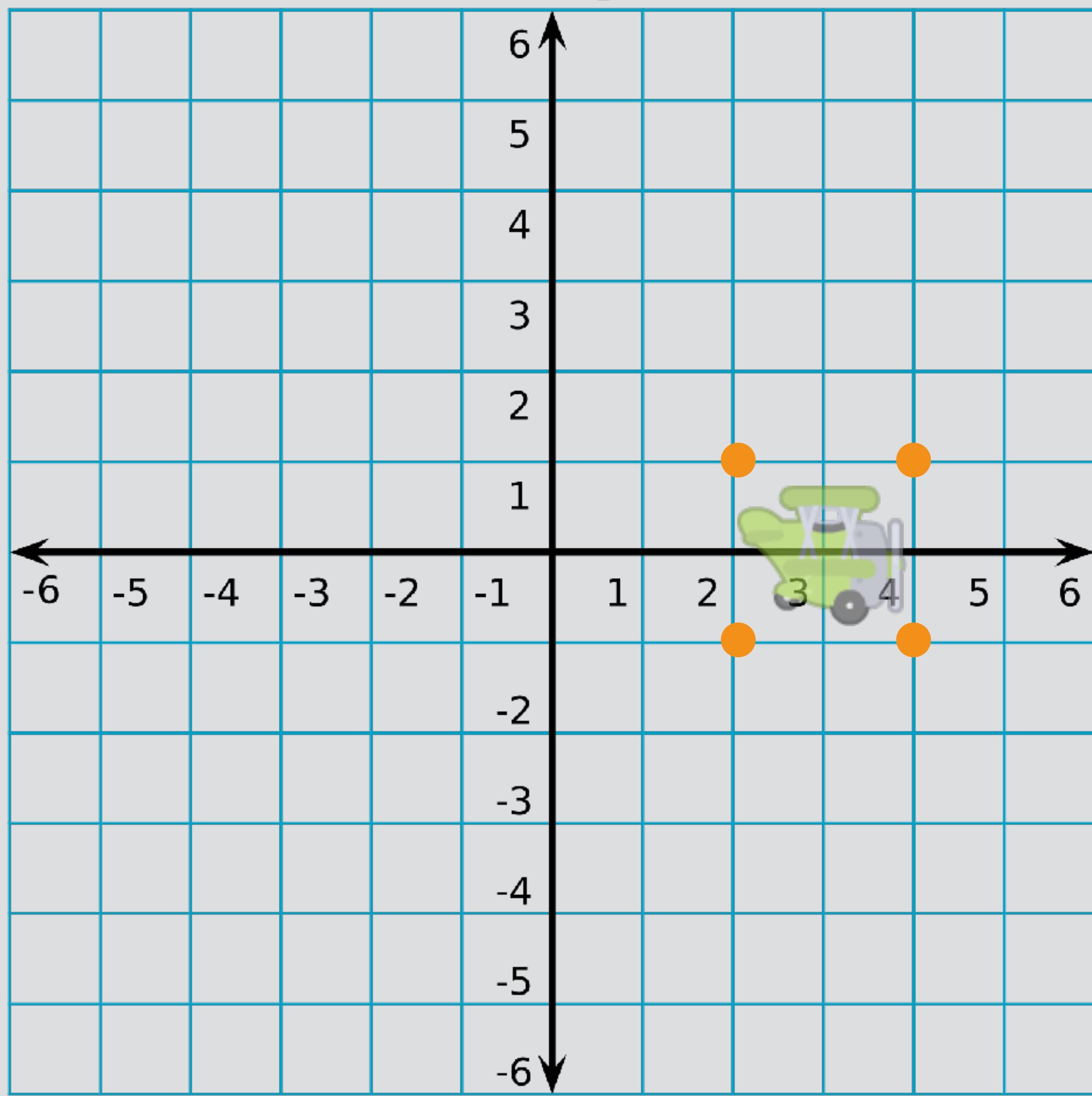
y-axis

x-axis



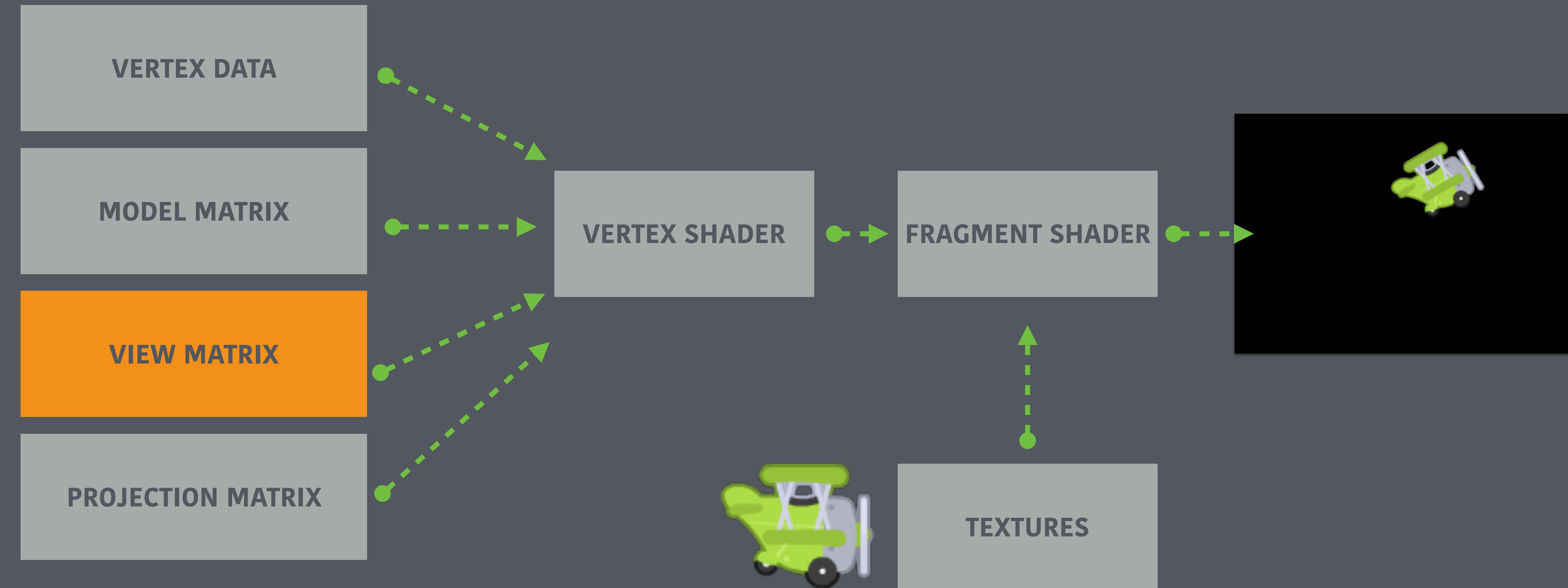
y-axis

x-axis



The model matrix is a **AFFINE TRANSFORMATION MATRIX**  
that defines the transform of each object you draw.

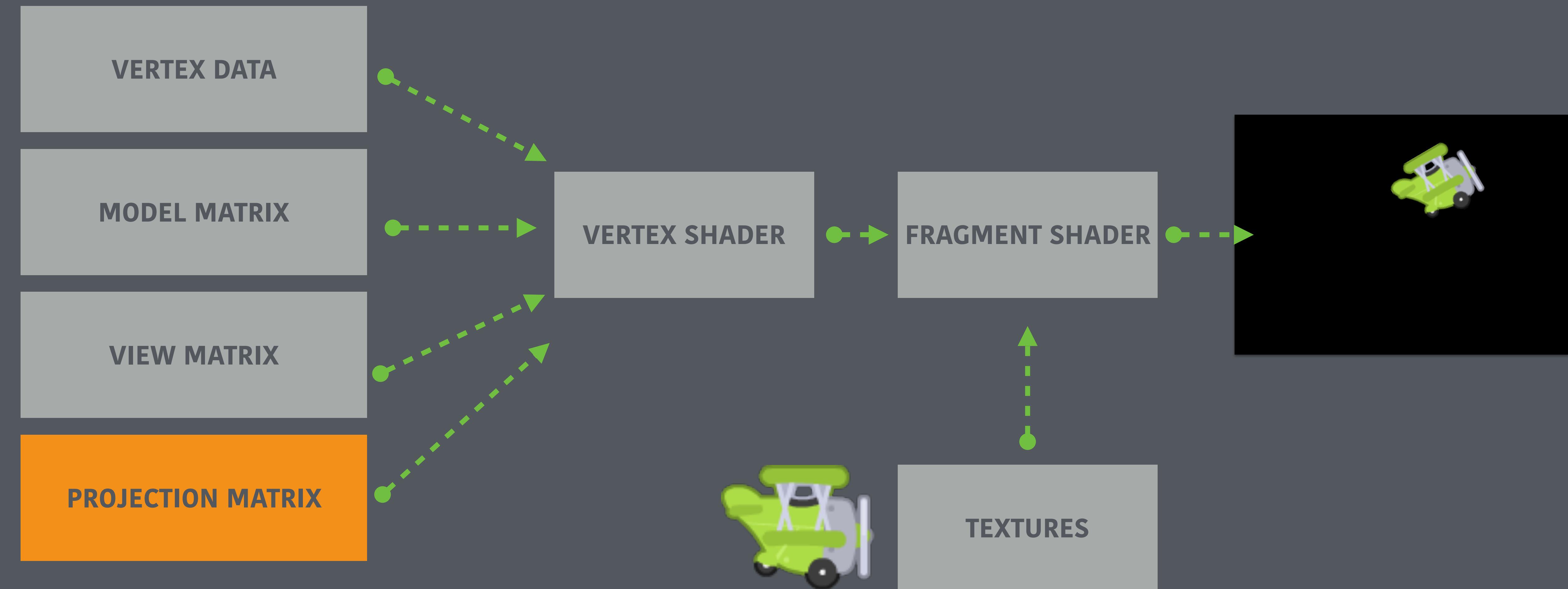
# The GPU pipeline



# View matrix

An affine transformation matrix that is applied to all objects.

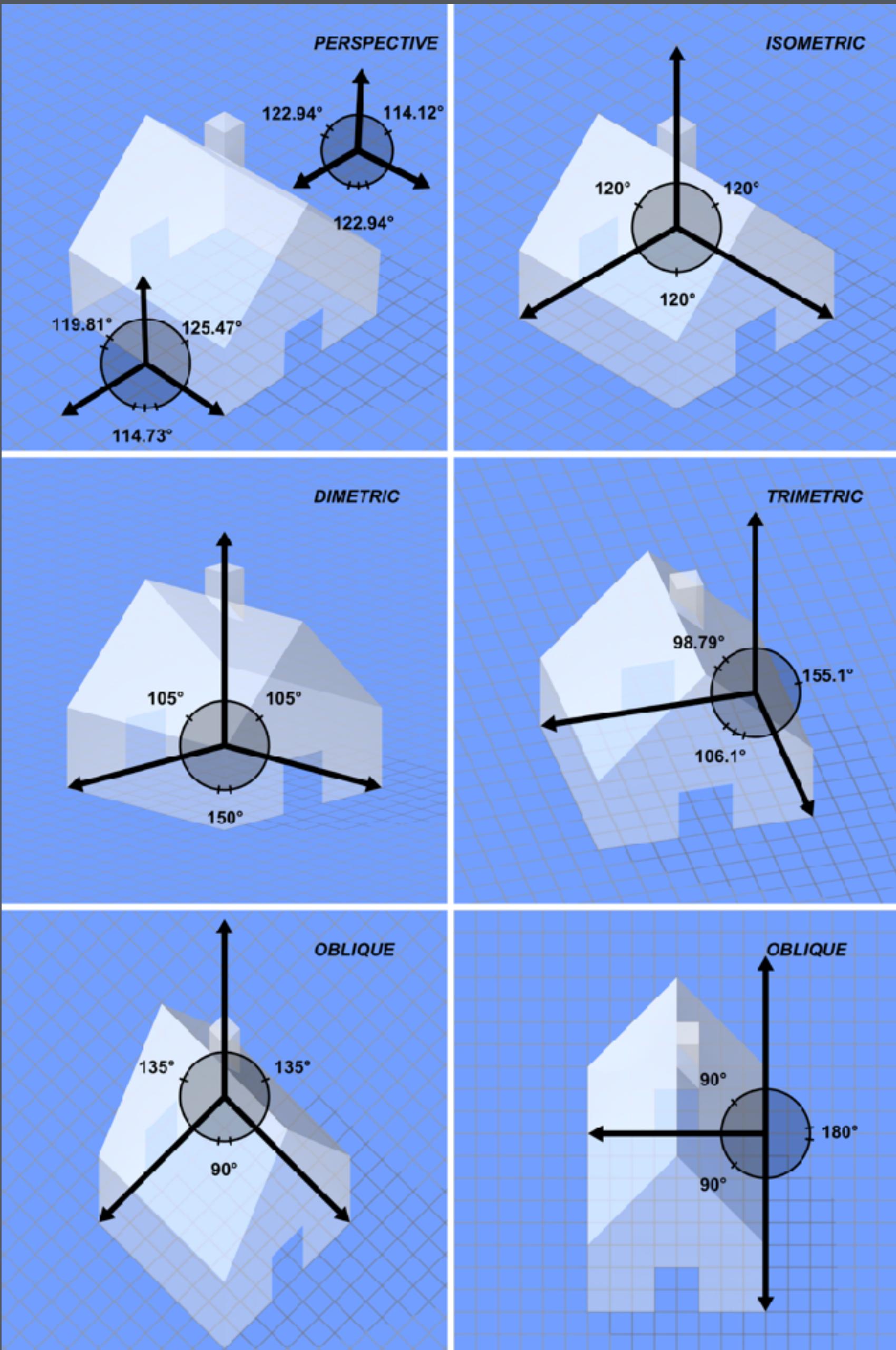
# The GPU pipeline



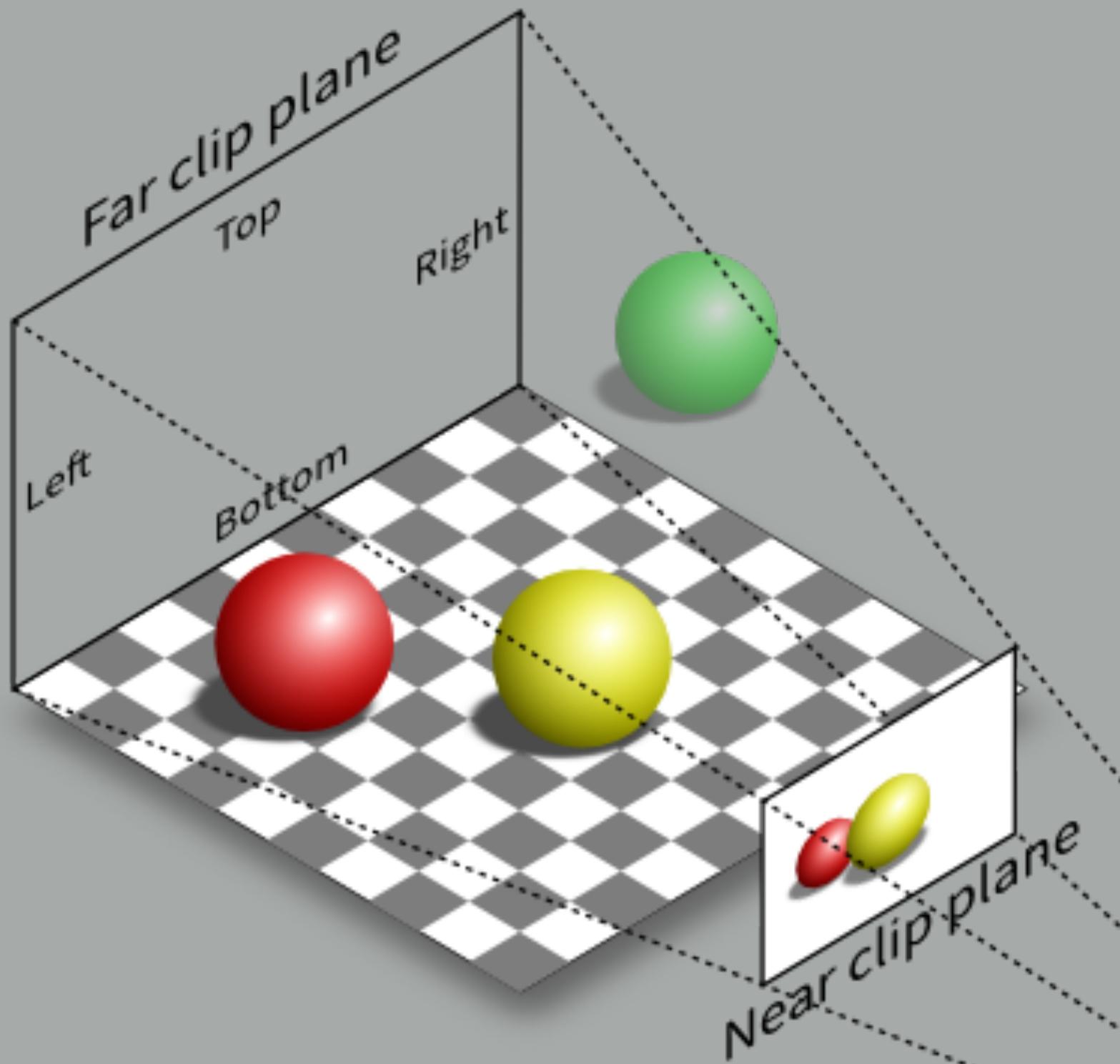
# Projection matrix

# What is projection?

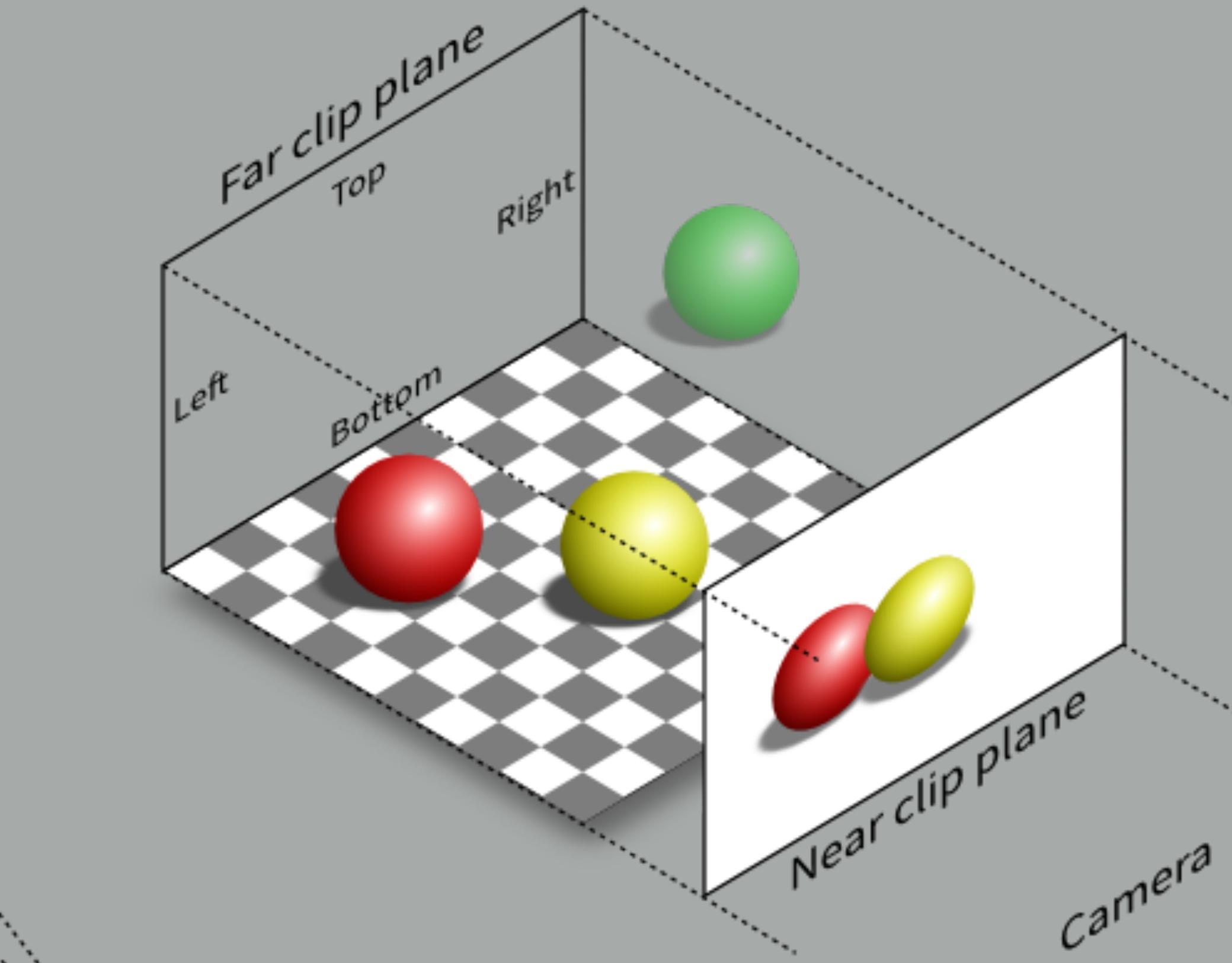
A means of representing a  
three-dimensional  
object in two-dimensions.



# Perspective vs. Orthographic projection

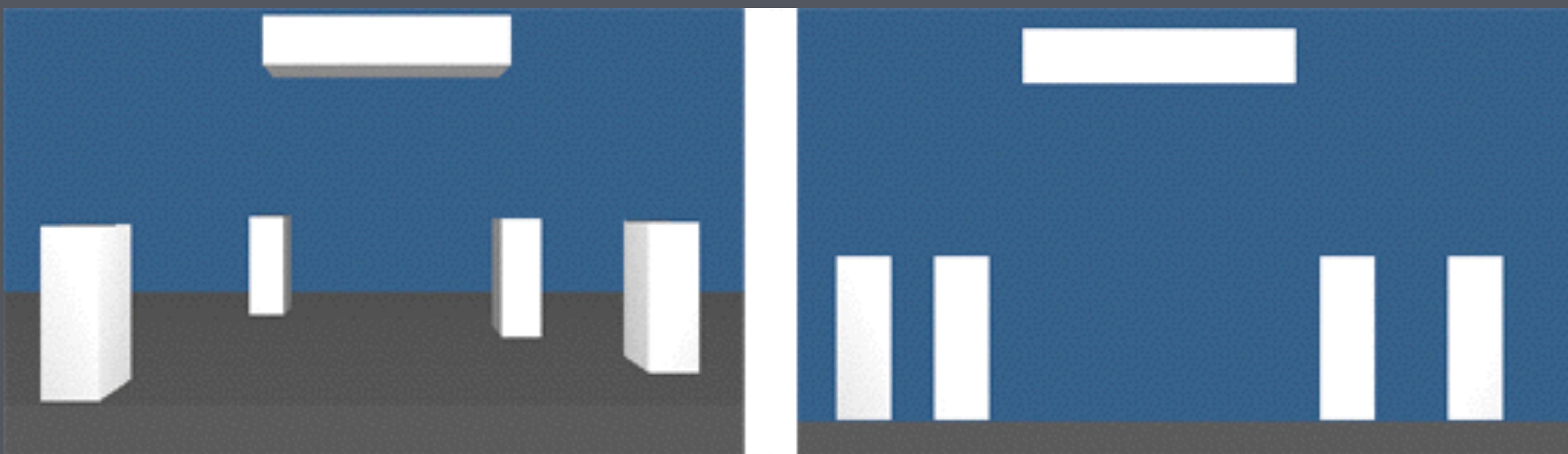
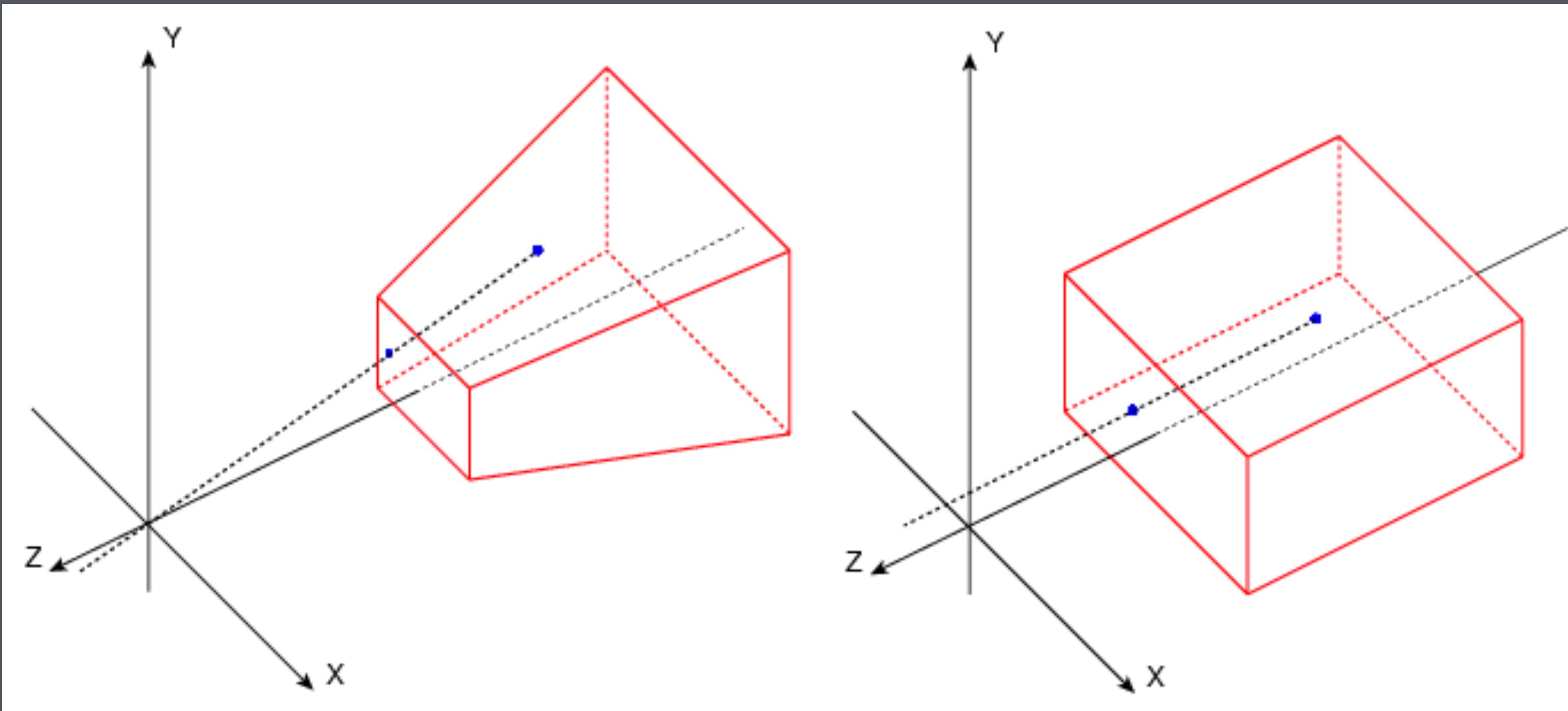


Perspective projection (P)

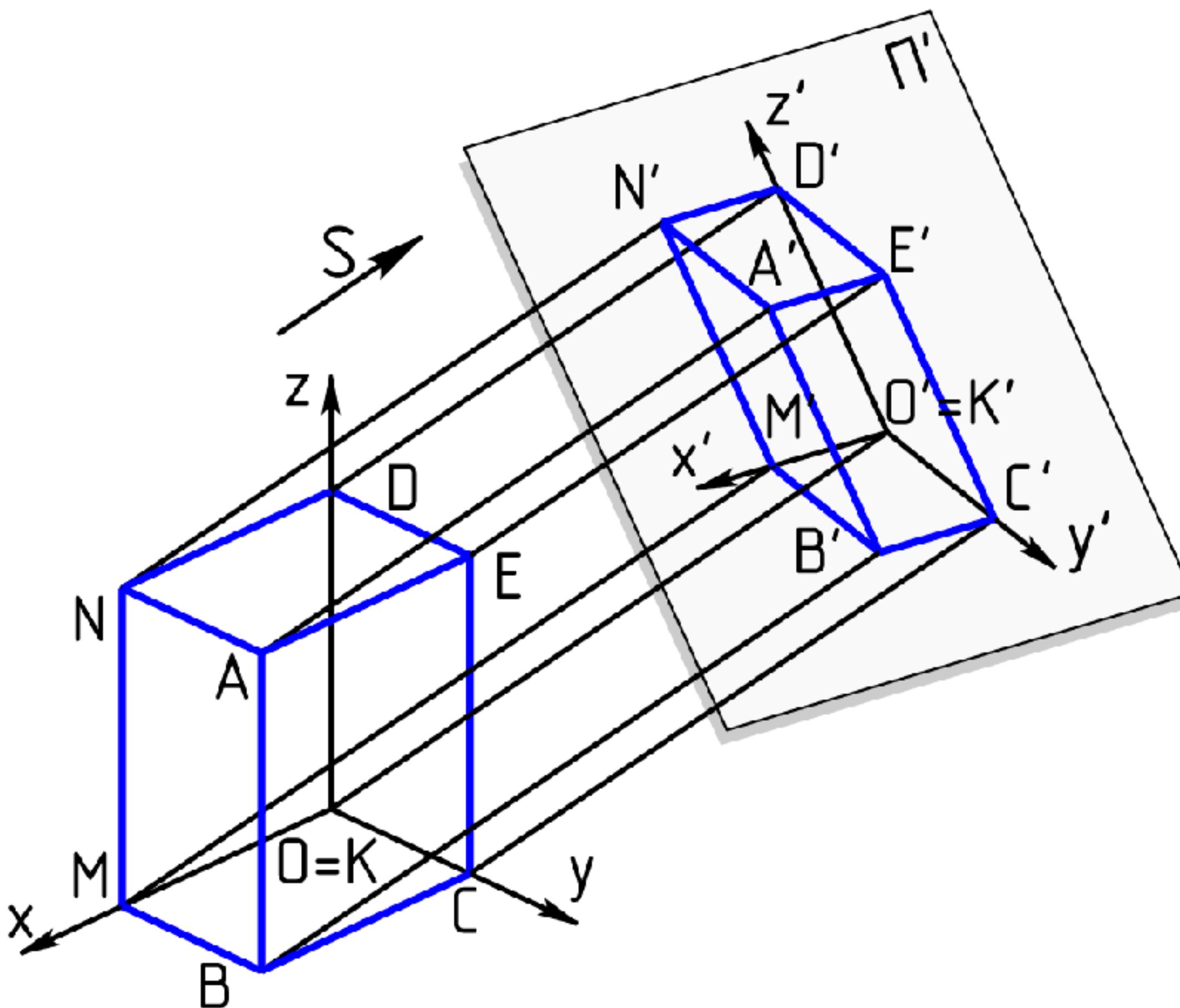


Orthographic projection (O)

# Perspective vs. Orthographic projection

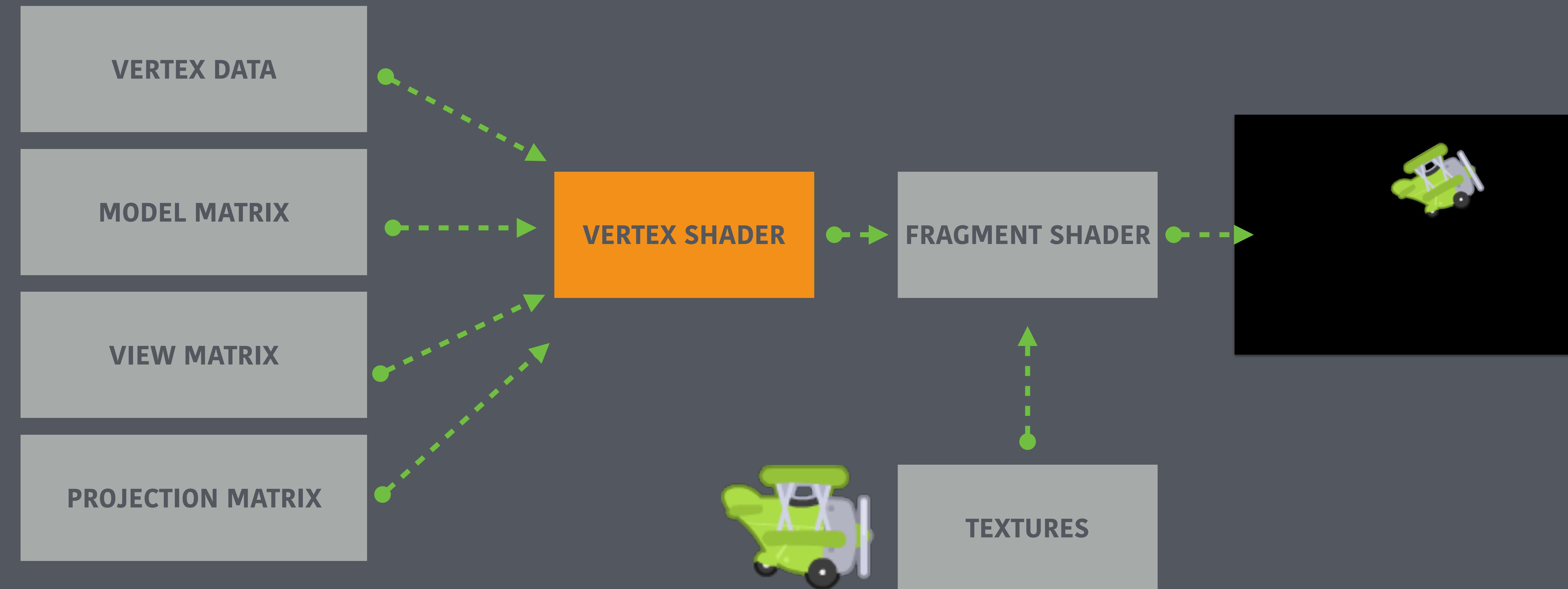


# Projection matrix



Vertex position \* projection matrix = screen vertex position

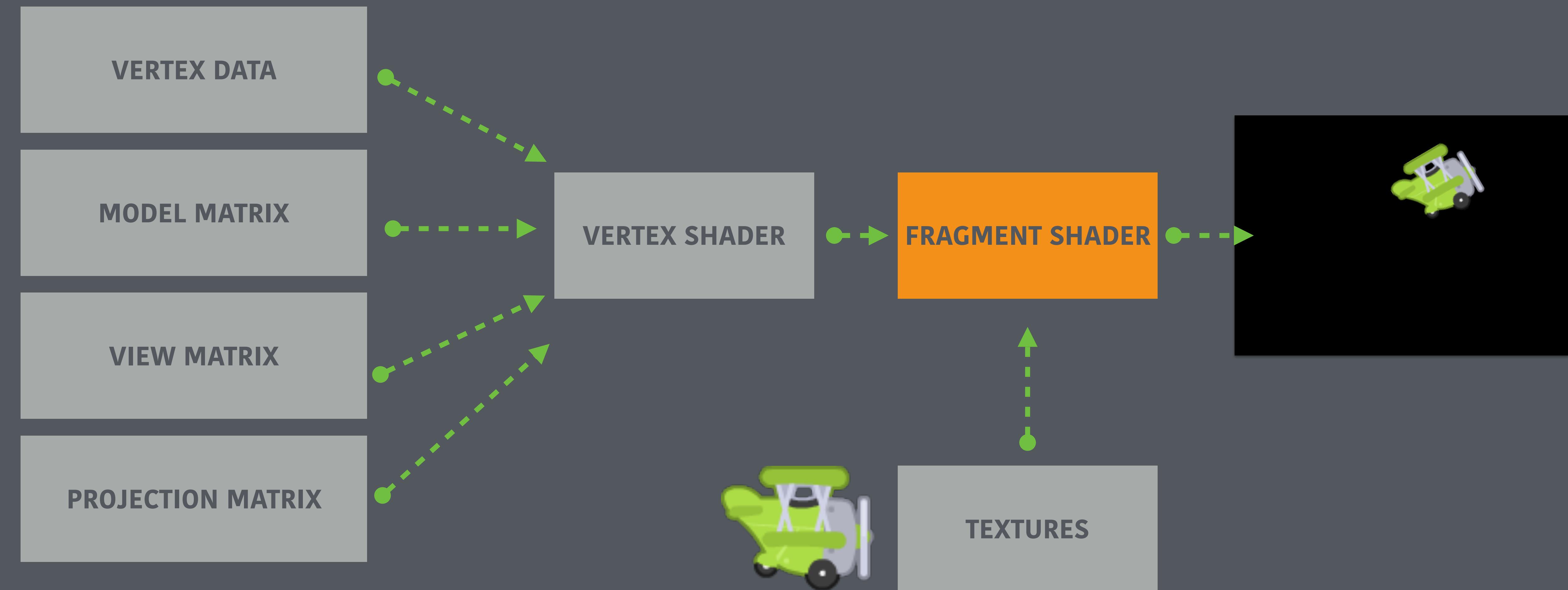
# The GPU pipeline



# The vertex shader

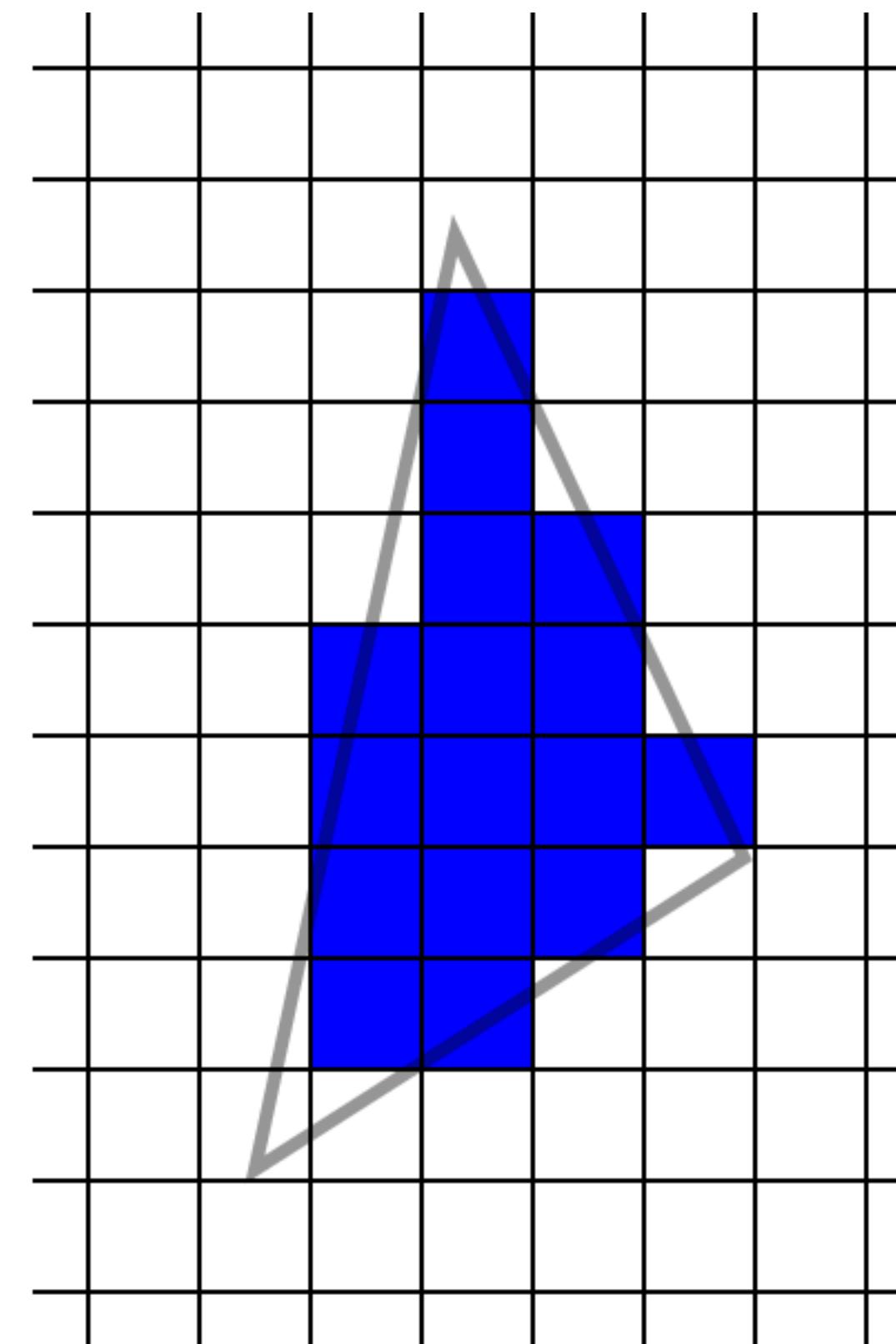
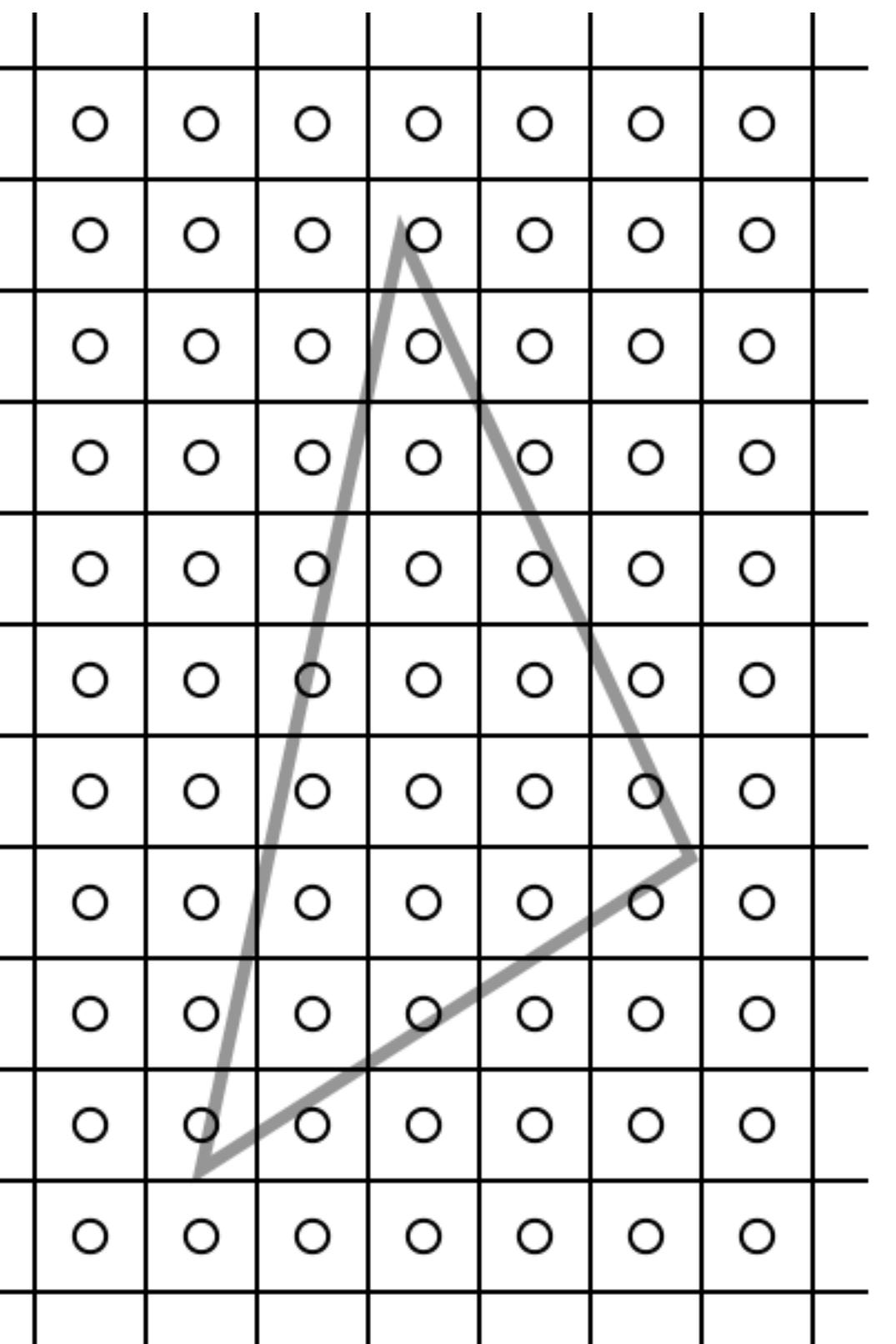
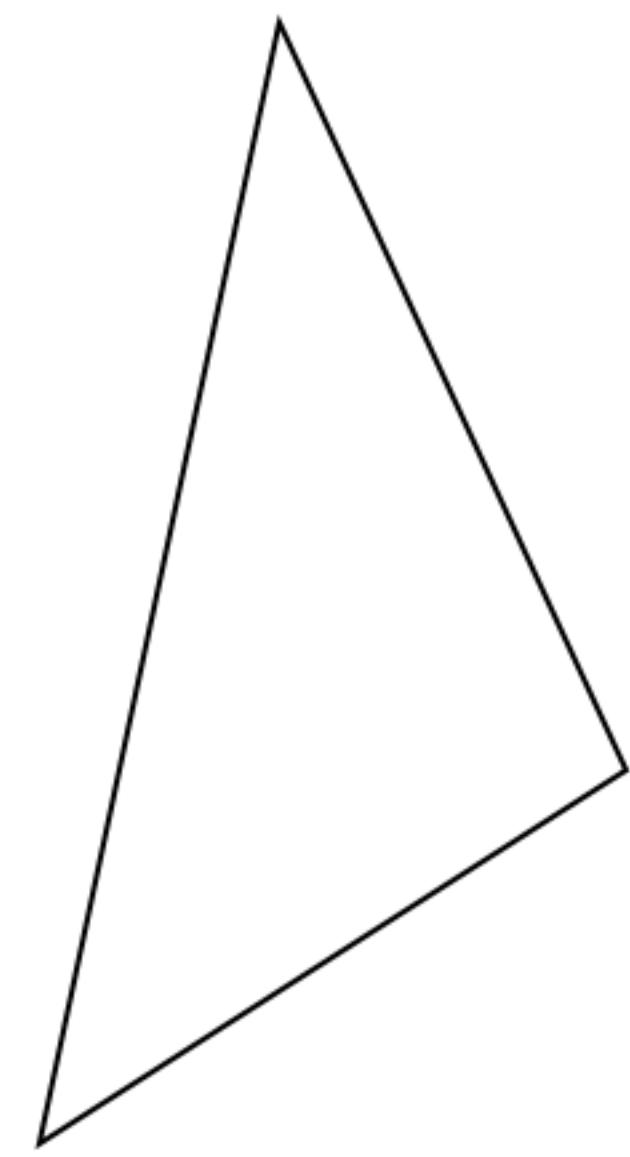
A program that transforms the attributes  
(such as position, color or others)  
of every vertex passed to the GPU.

# The GPU pipeline

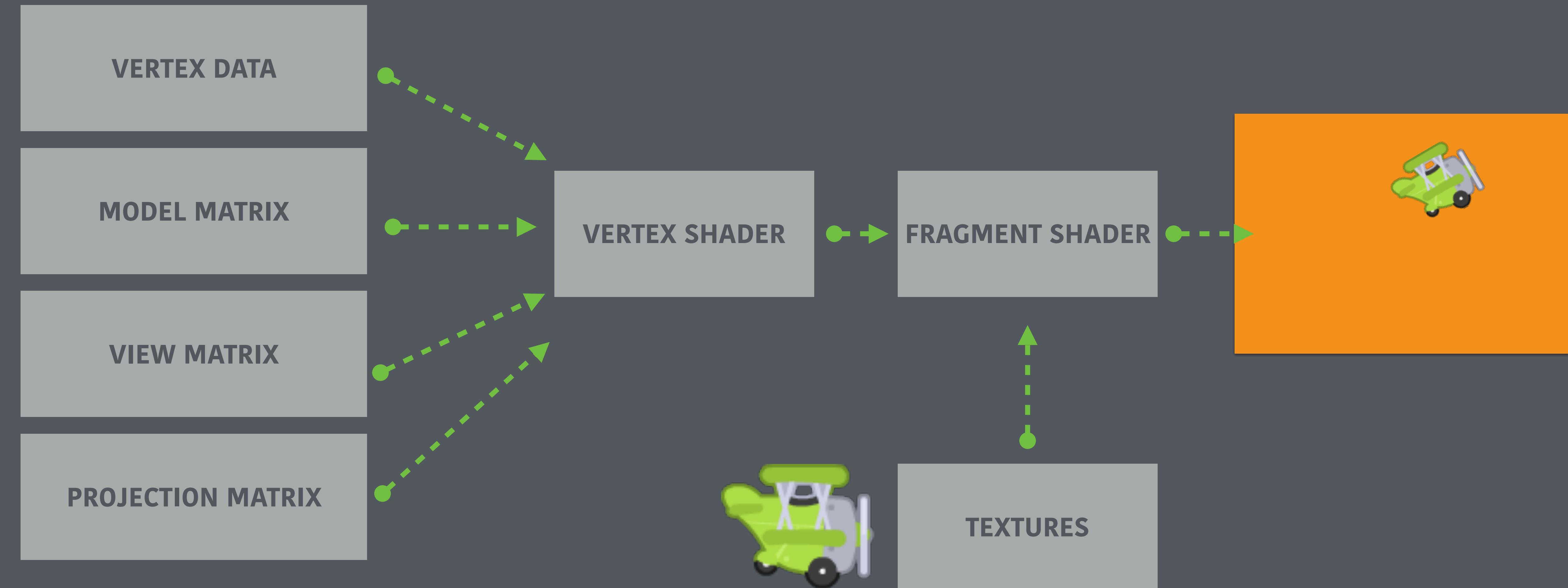


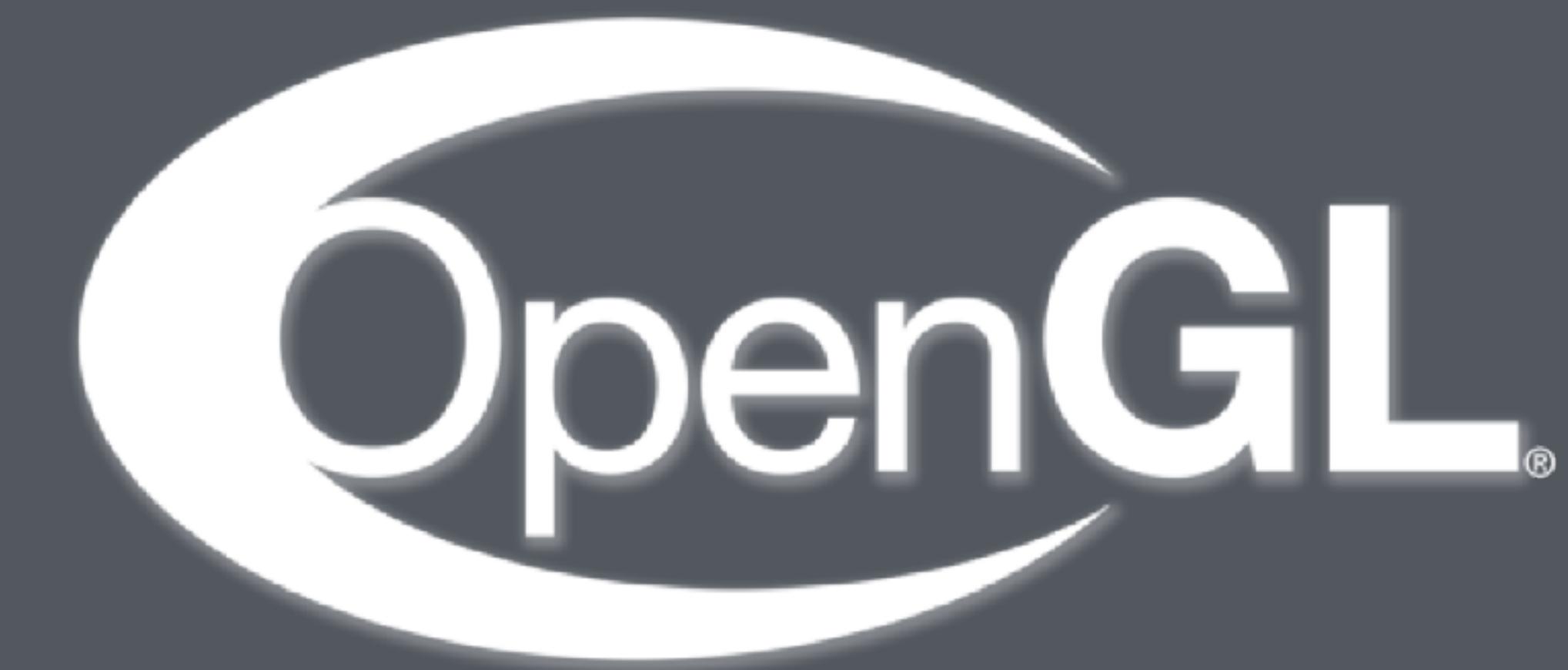
# The fragment shader

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



# The GPU pipeline



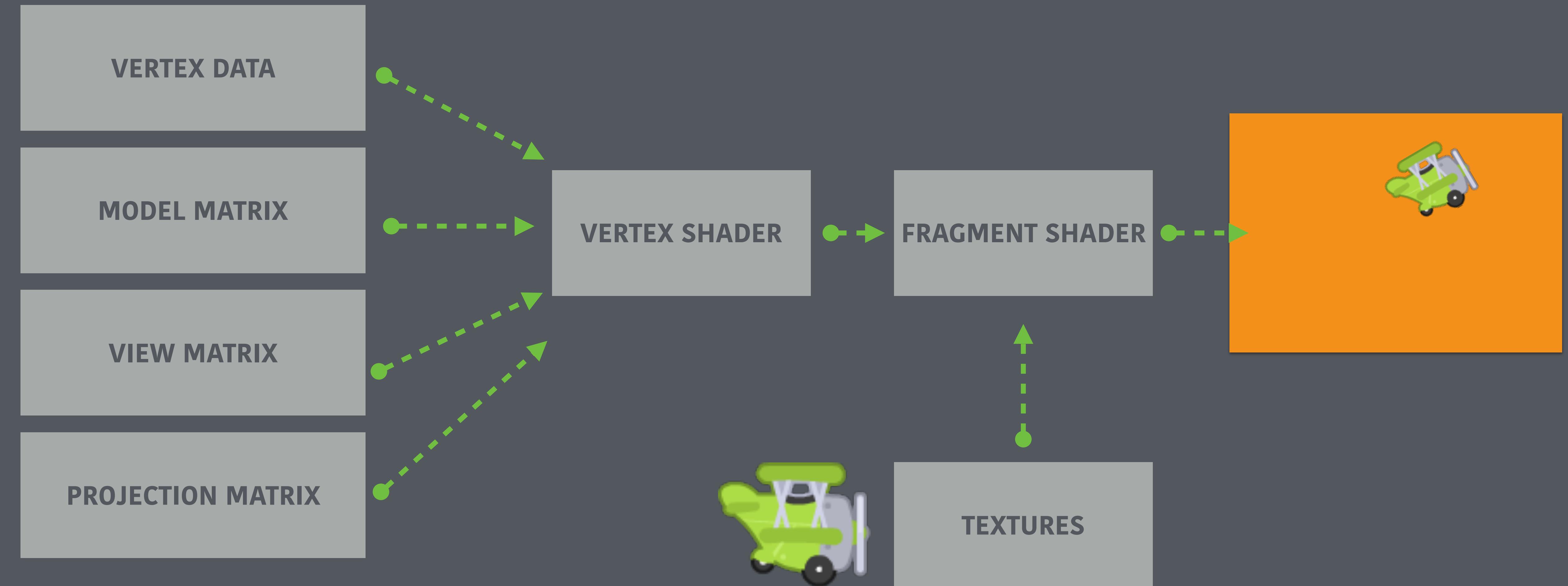


The setup  
(happens only once!)

```
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);
```

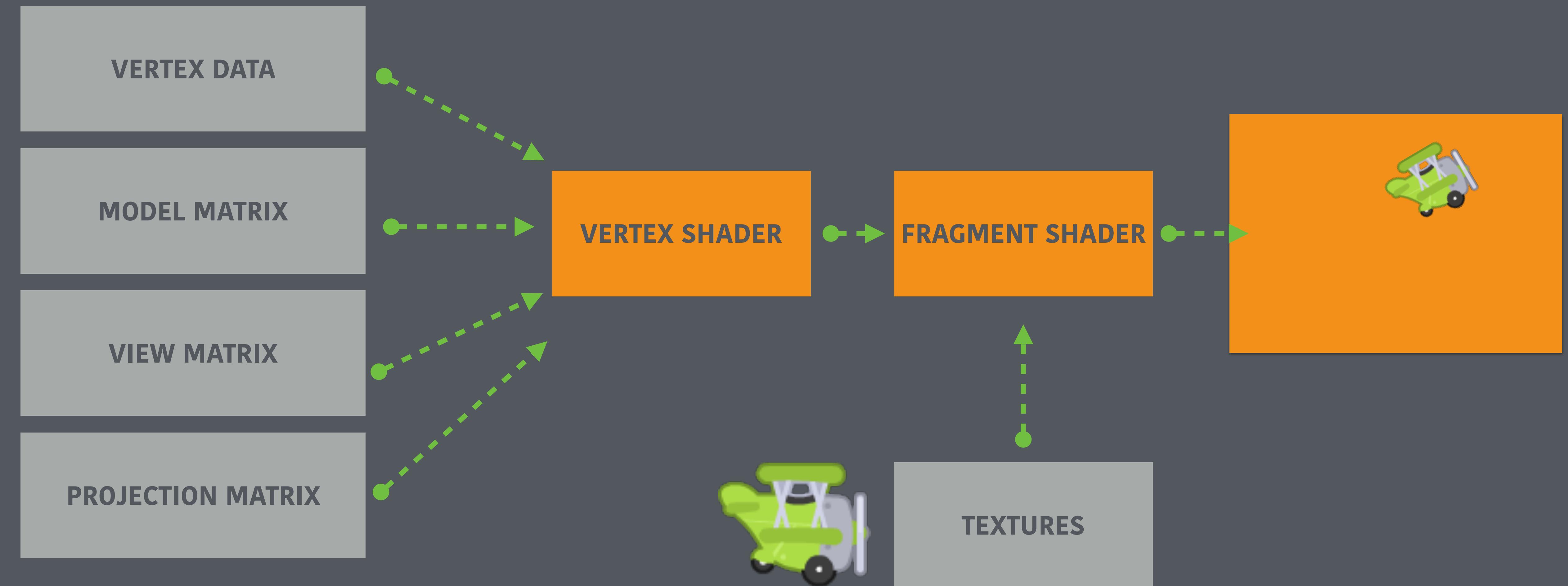


# The **ShaderProgram** class.

```
#include "ShaderProgram.h"

// ...

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



# The **Matrix** class.

```
#include "Matrix.h"

// ...

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

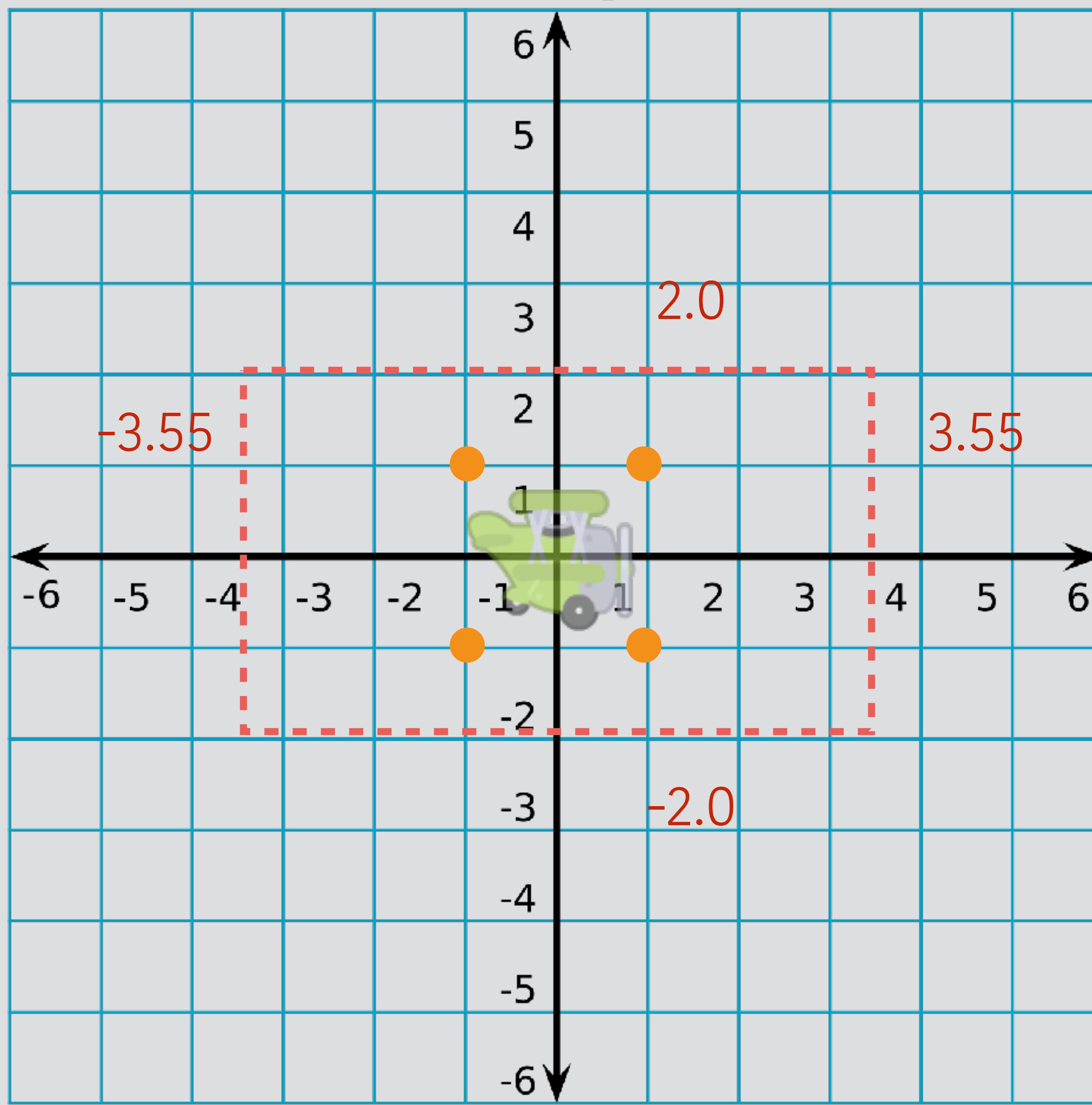
```
void Matrix::setOrthoProjection (float left, float right,  
float bottom, float top, float near, float far);
```

Sets an orthographic projection in a matrix.

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

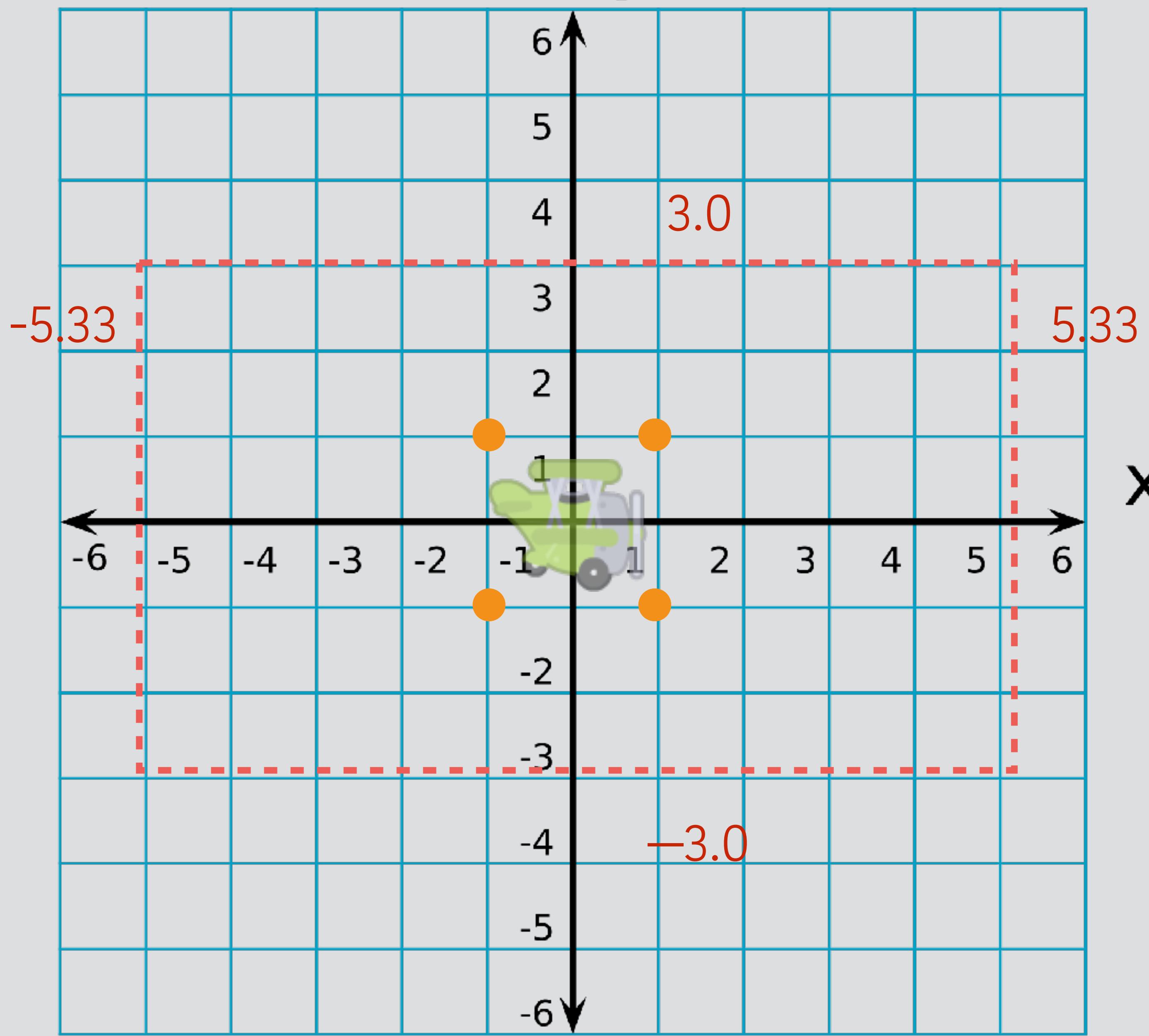
y-axis

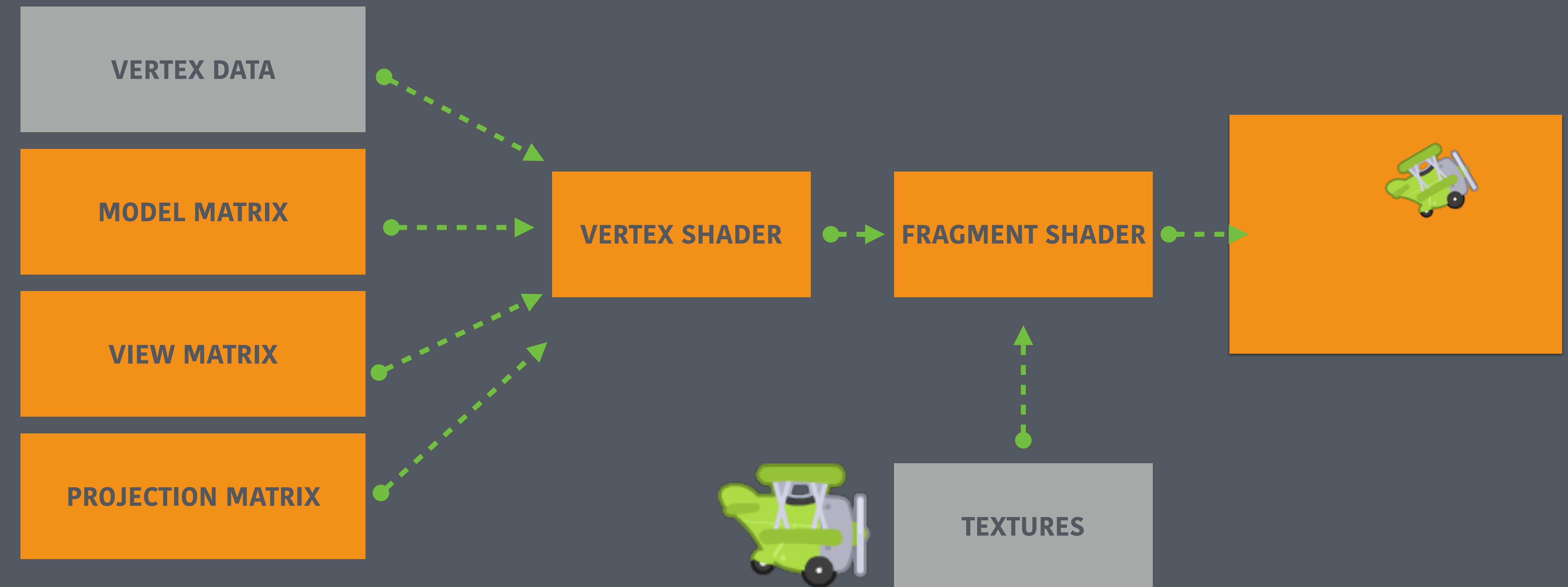
x-axis



y-axis

x-axis





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);
```

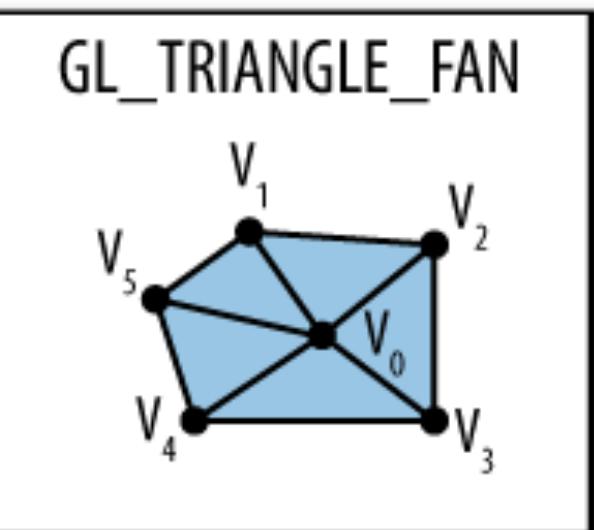
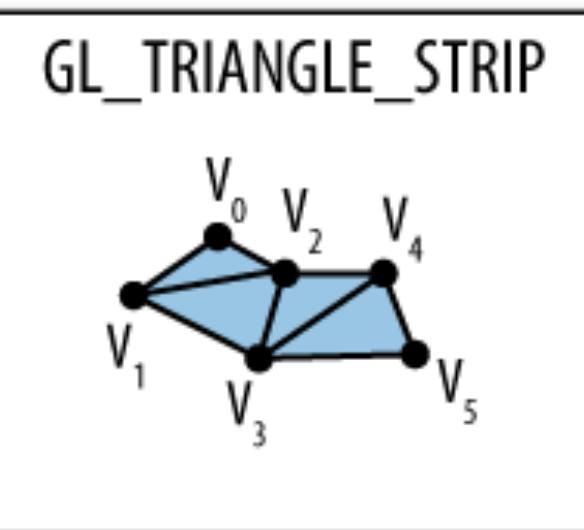
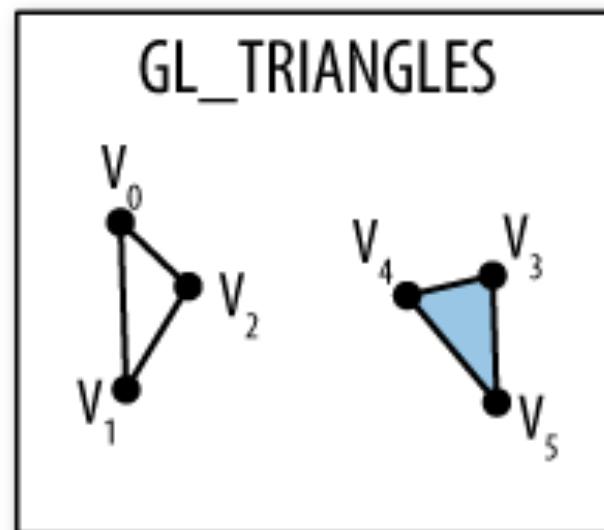
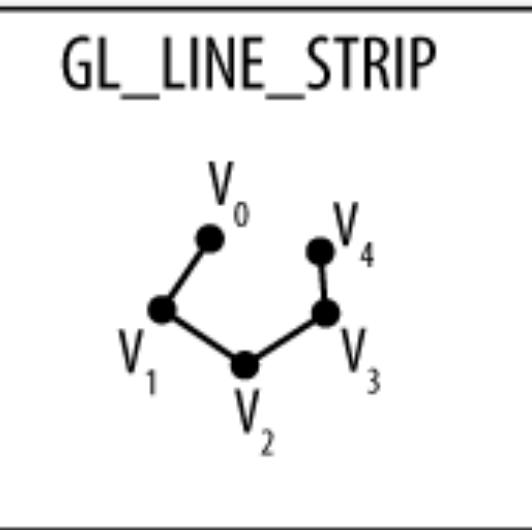
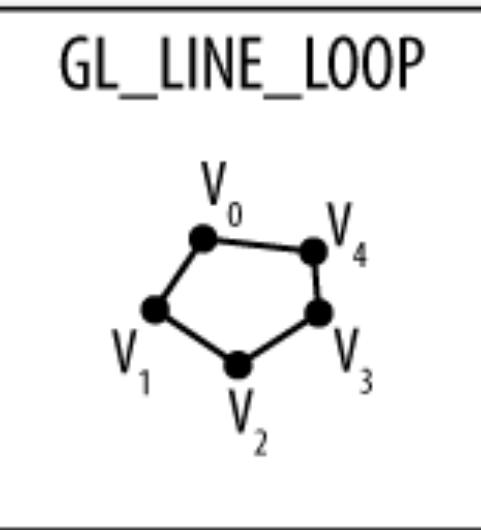
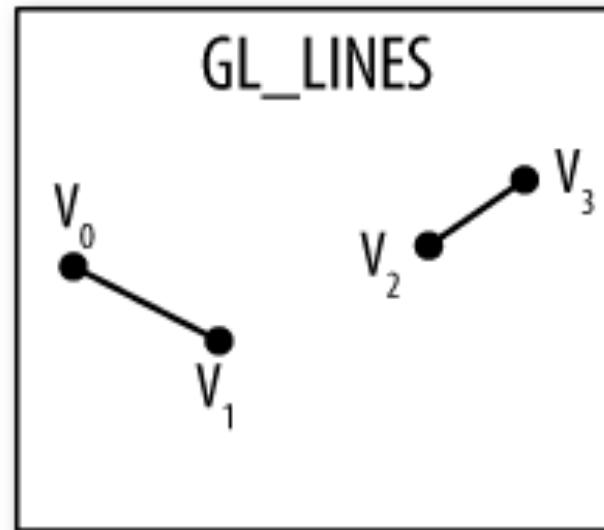
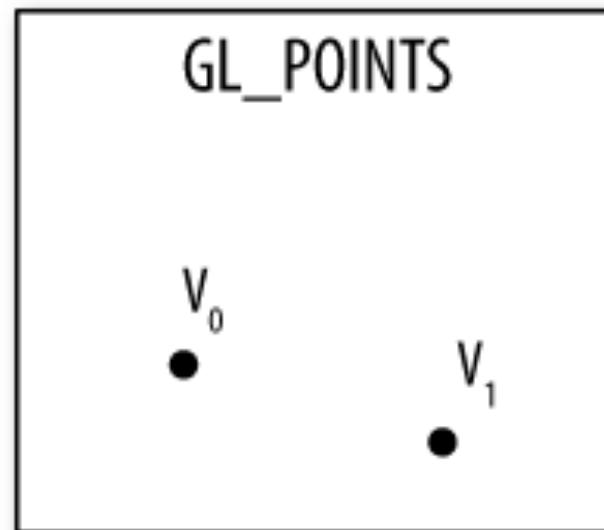
Enables a **vertex attribute**.

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

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

Render previously defined **arrays**.

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

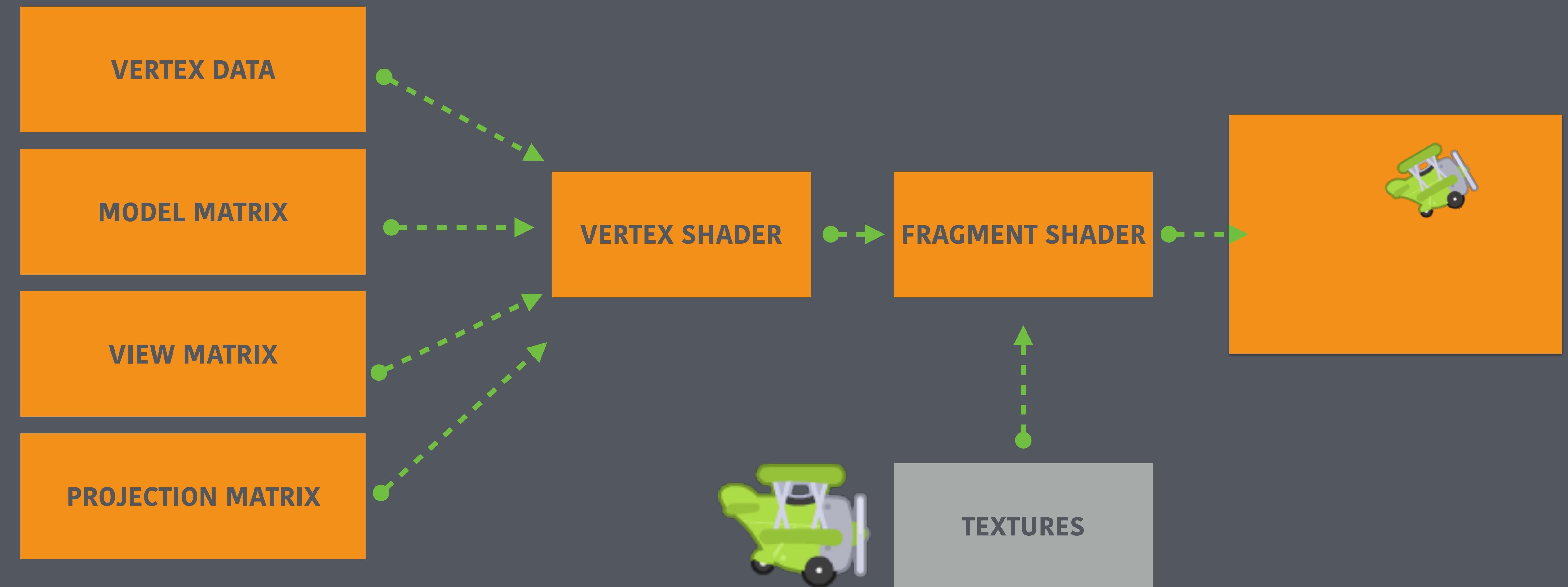


```
void glDisableVertexAttribArray (GLuint index);
```

Disables a **vertex attribute array**.

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

Putting it all together.



# Setup (before the loop)

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

ShaderProgram program(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);

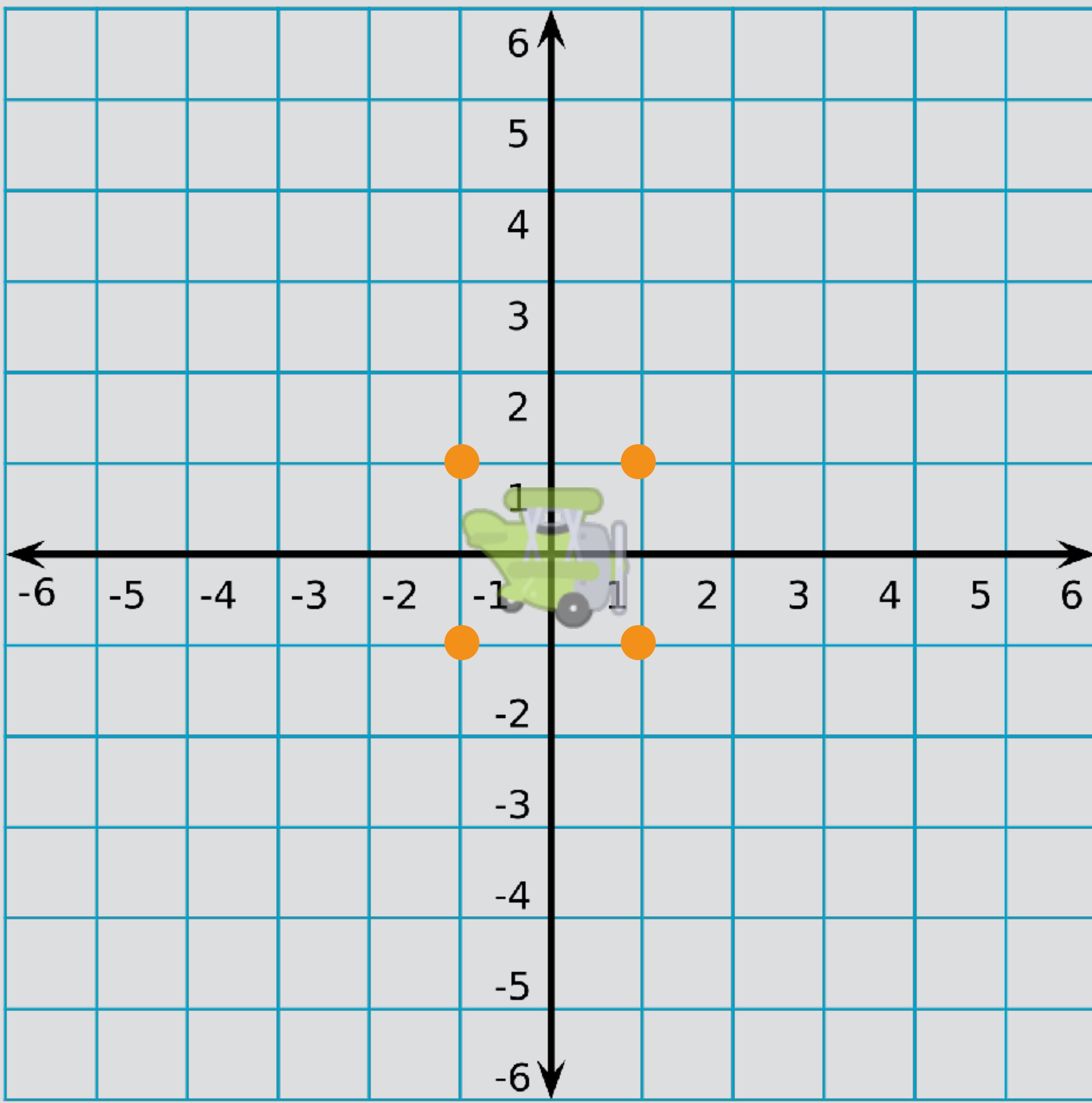
SDL_GL_SwapWindow(displayWindow);
```

# Transformations

# Identity matrix.

y-axis

x-axis



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

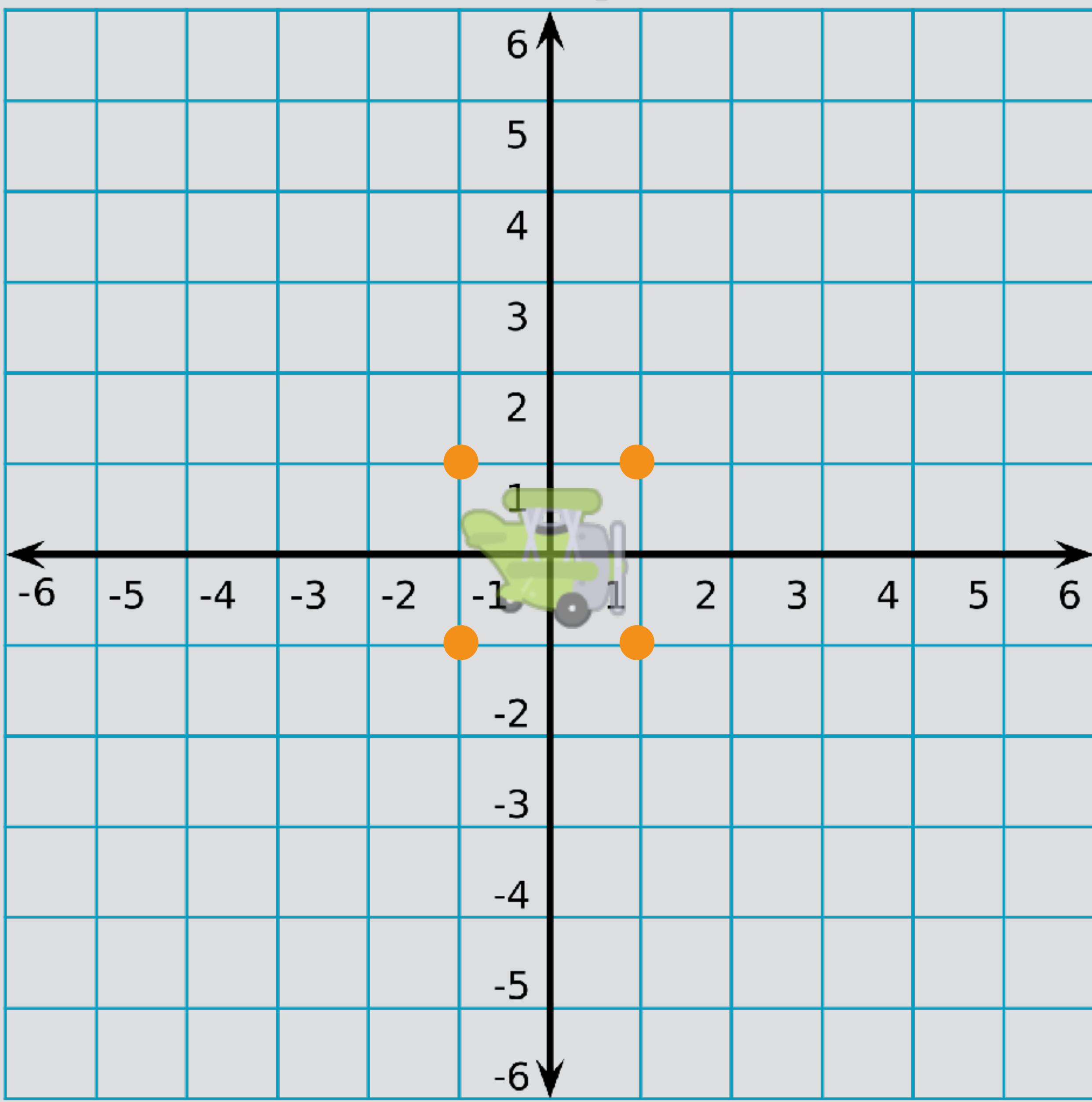
**Resets** a matrix to have no transformations.

```
Matrix modelMatrix;  
modelMatrix.identity();
```

# Translations

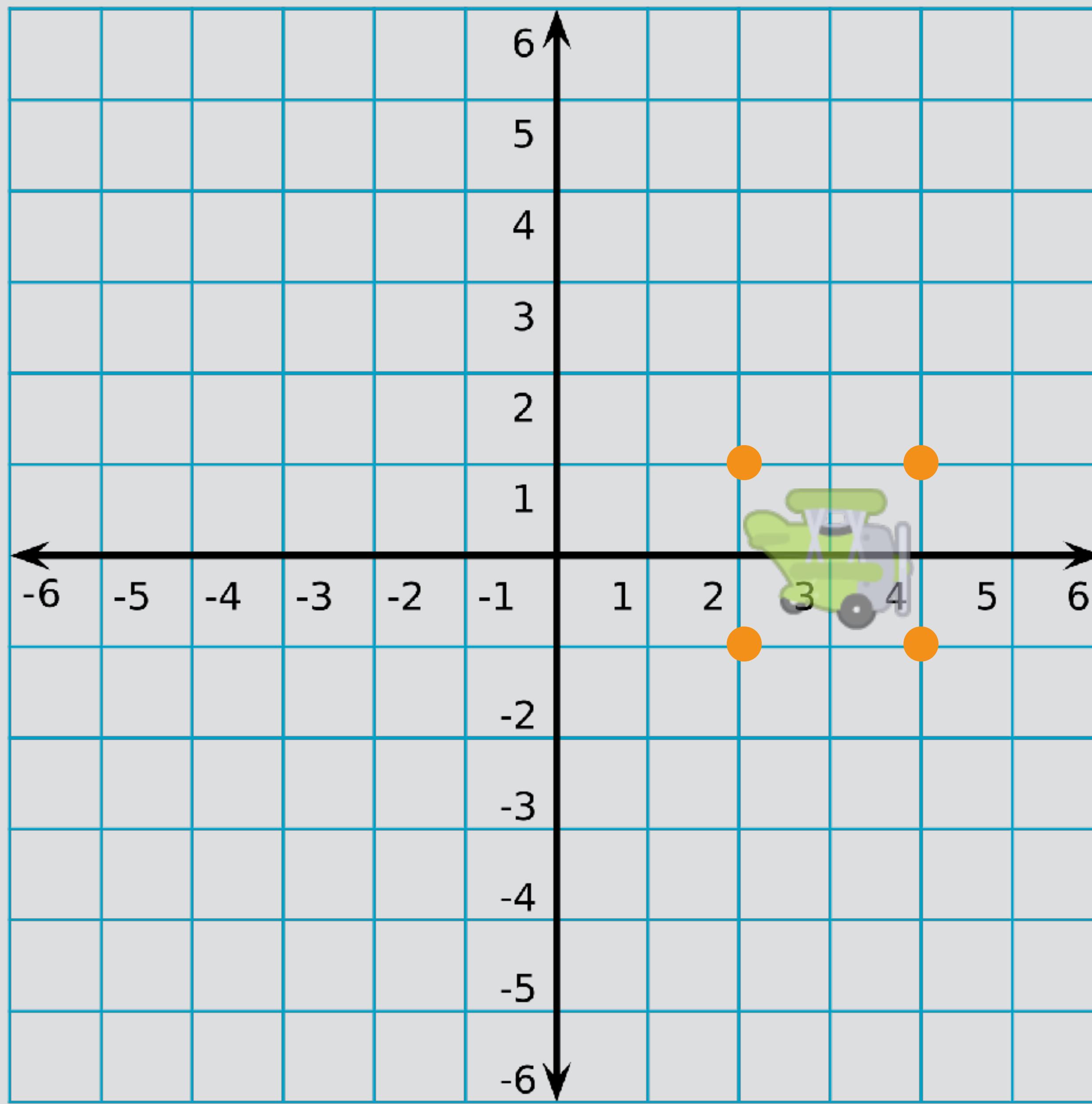
y-axis

x-axis



y-axis

x-axis



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

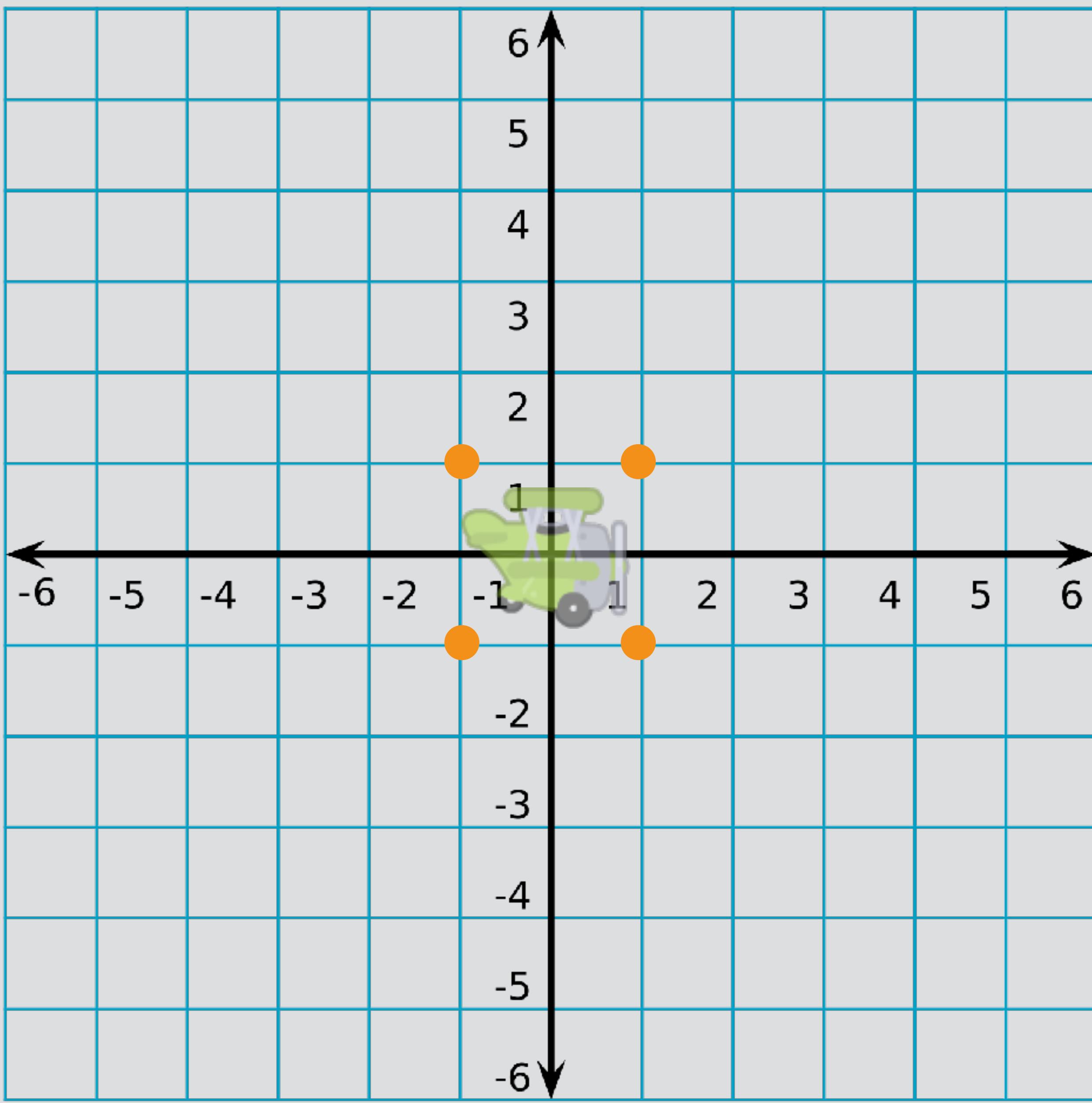
**Translates** a matrix by specified coordinates.

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

# Scaling

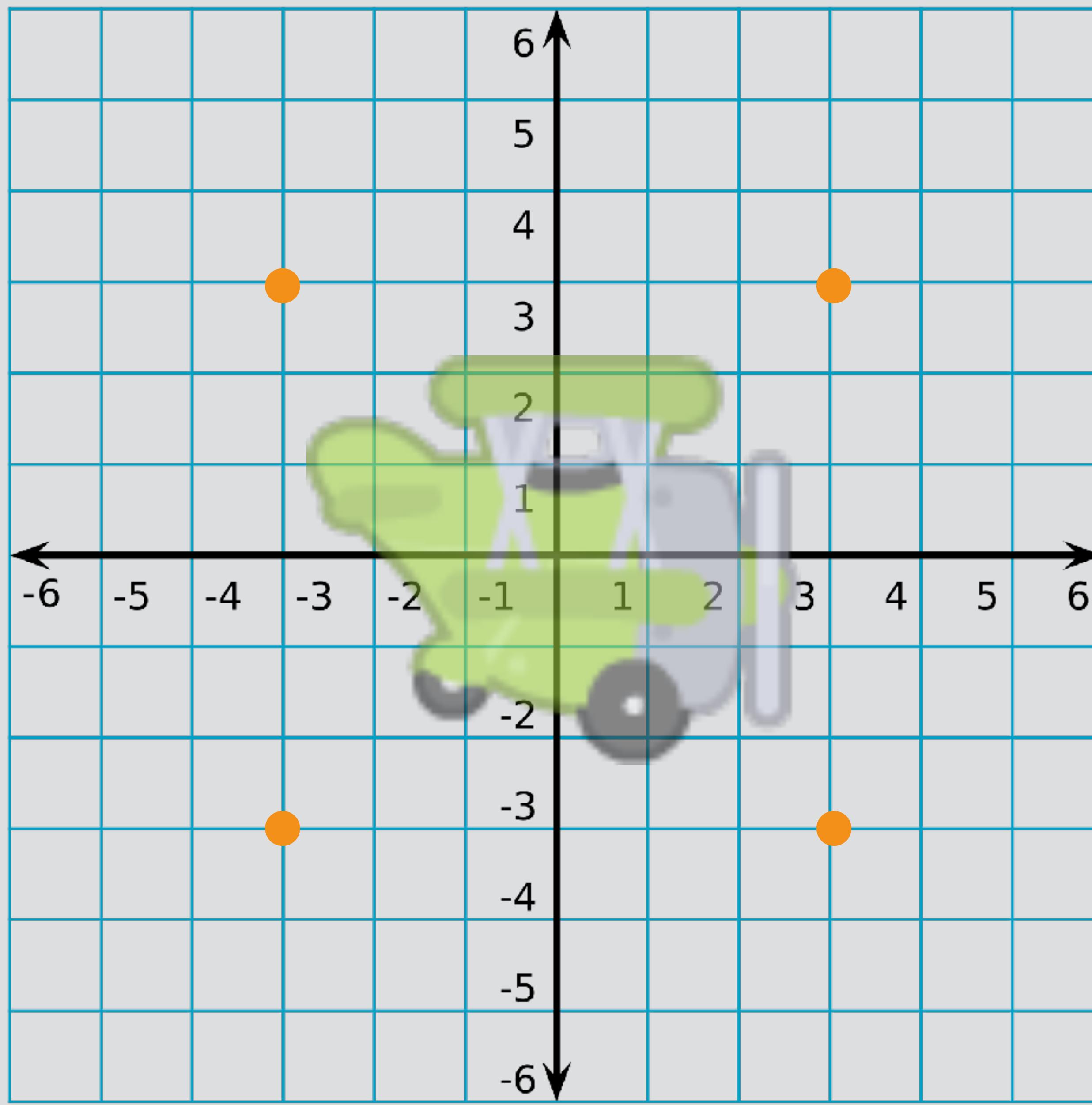
y-axis

x-axis



y-axis

x-axis



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

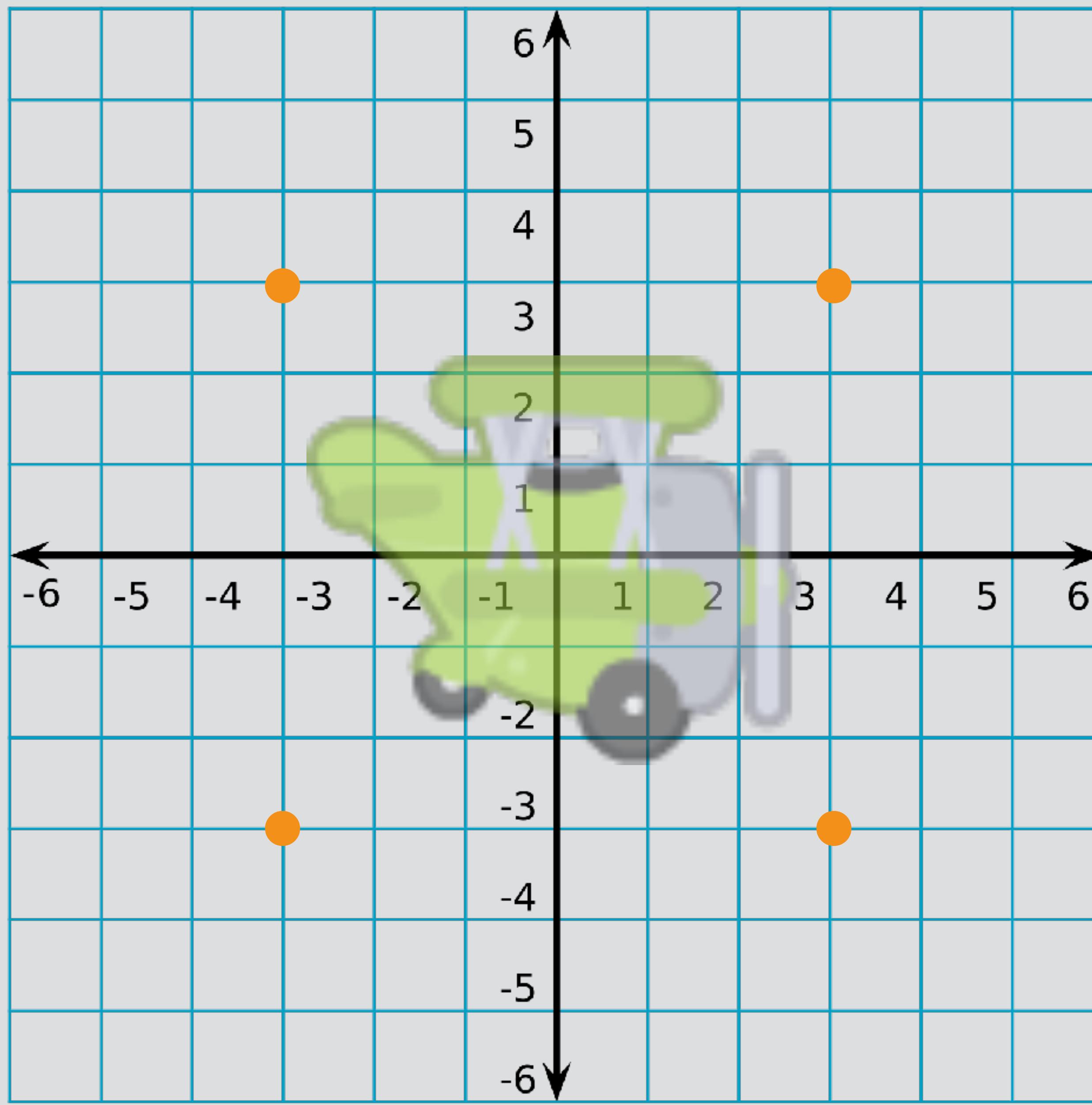
**Scales** a matrix by specified coordinates.

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

# Rotation

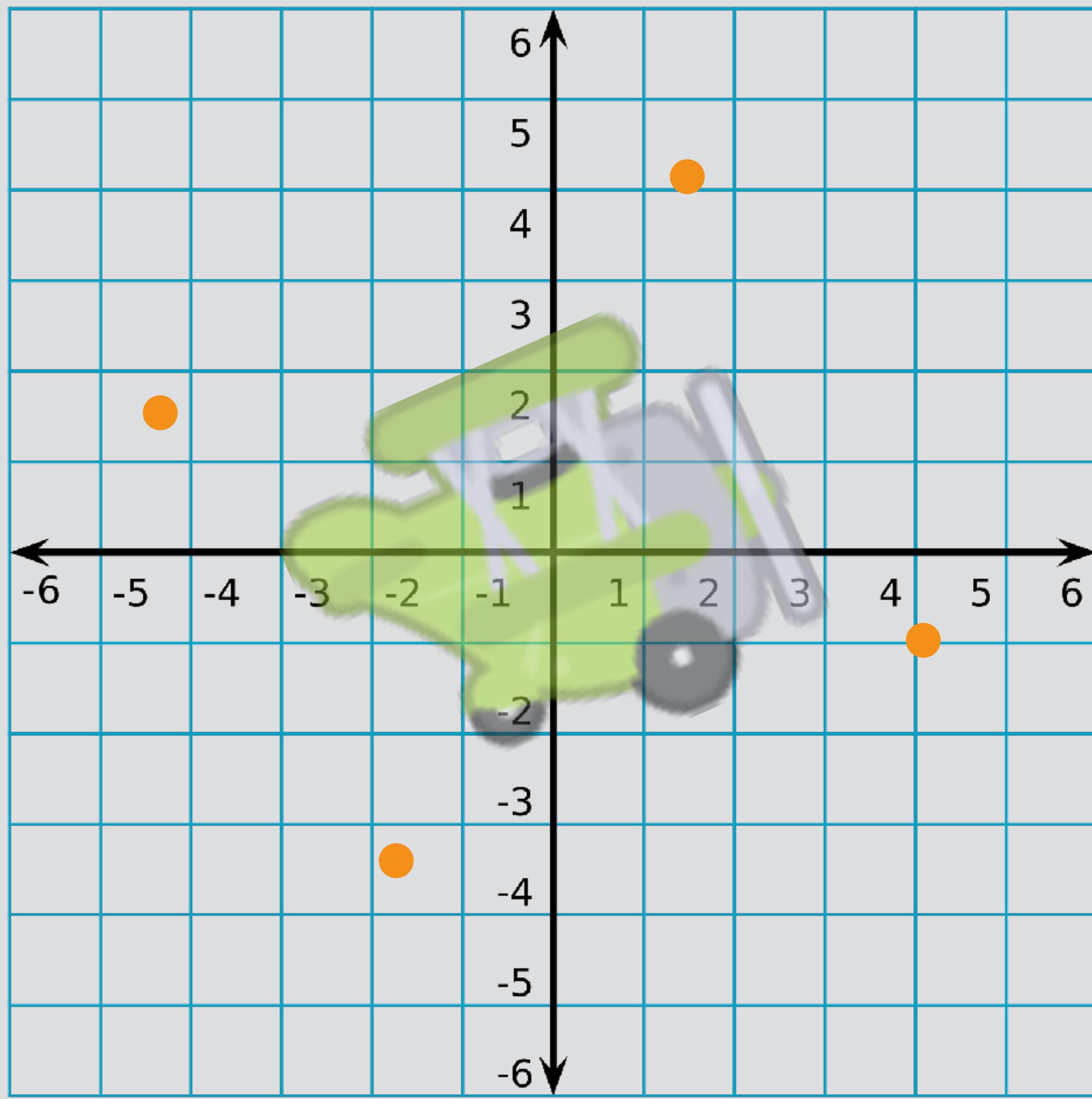
y-axis

x-axis



y-axis

x-axis



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

**Rotates** a matrix by specified radians.

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

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

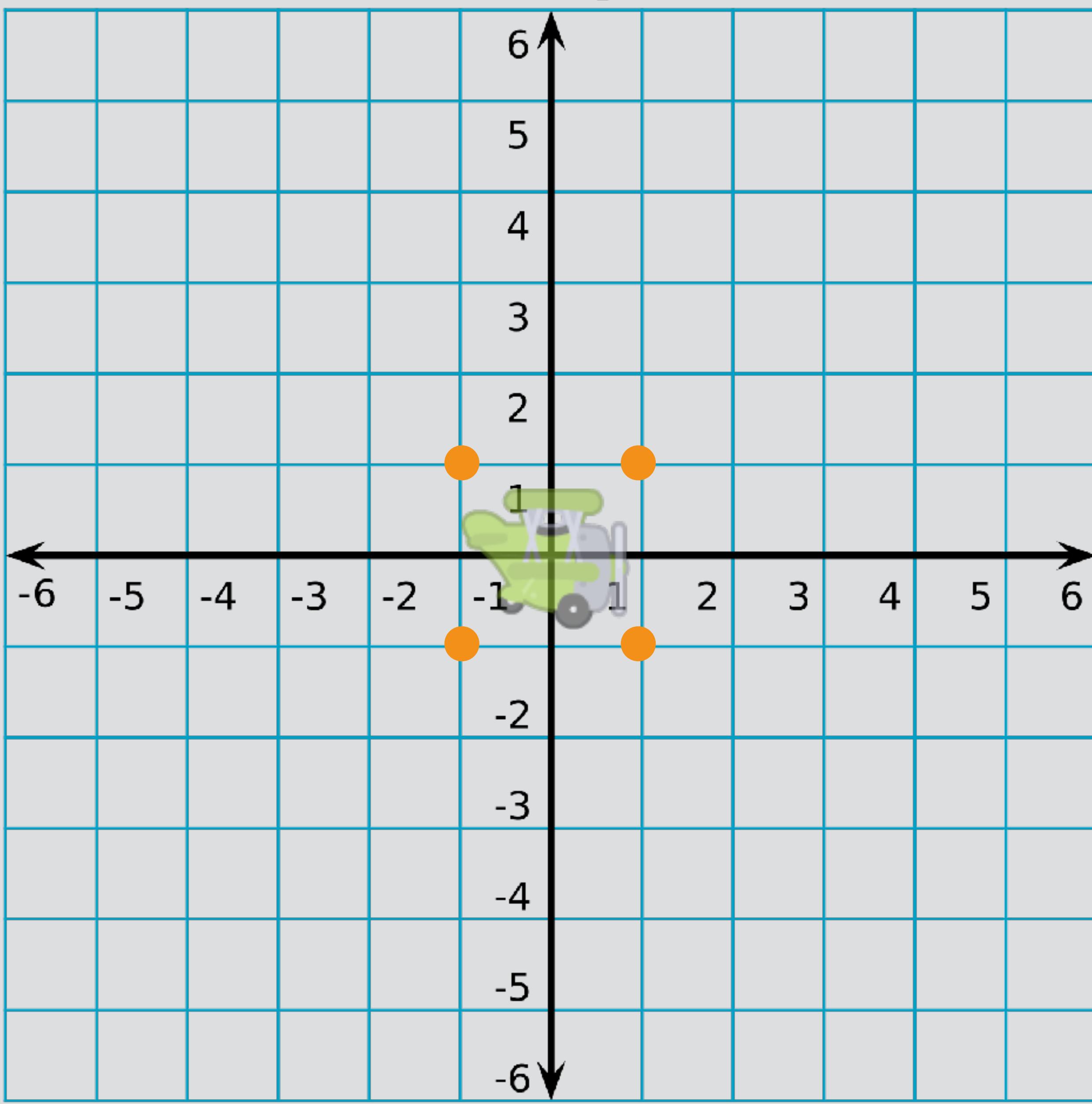
$$\text{RADIAN} = \text{DEGREE} * (\text{PI} / 180)$$

Matrix transformations  
are additive!

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

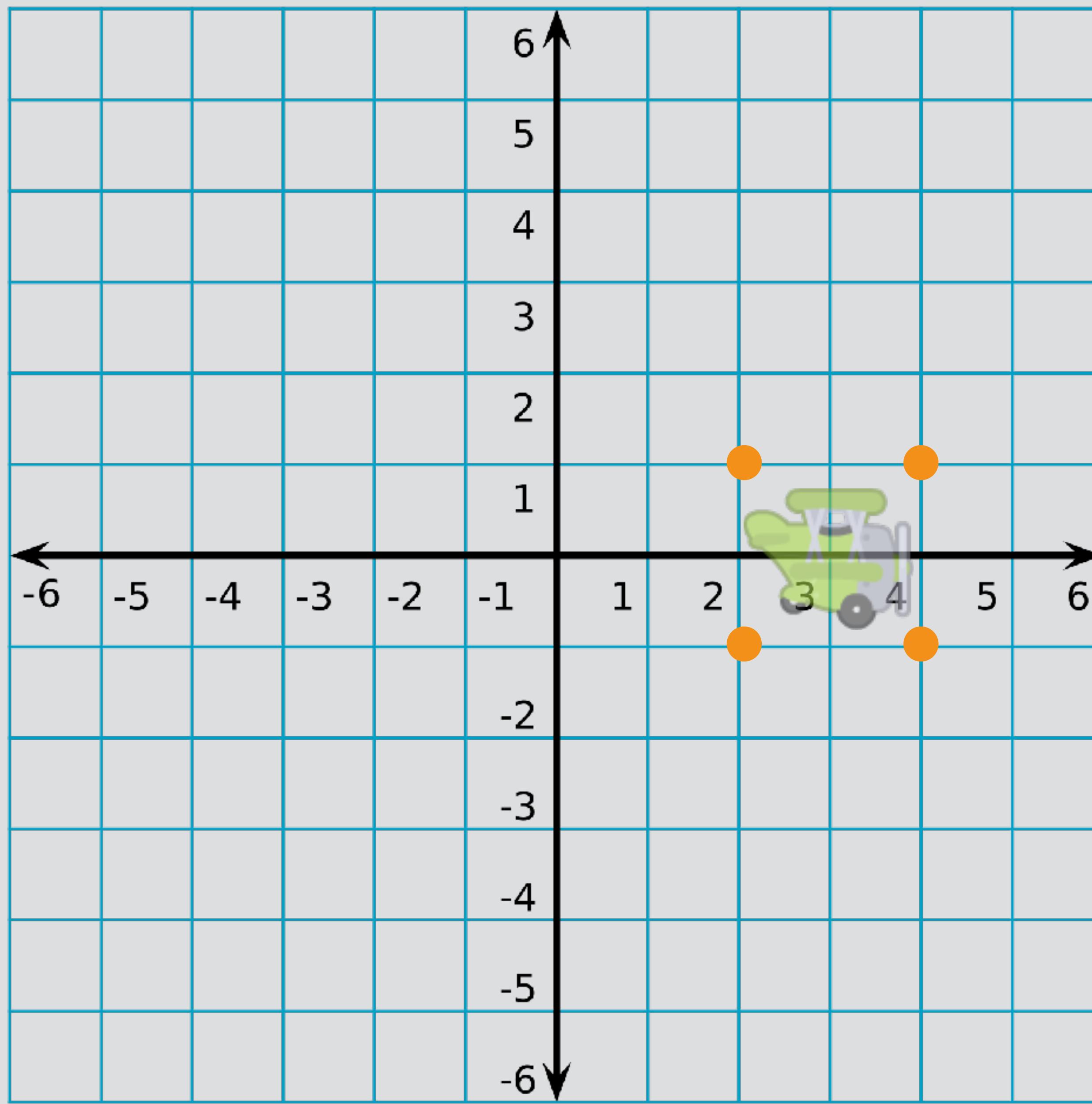
y-axis

x-axis



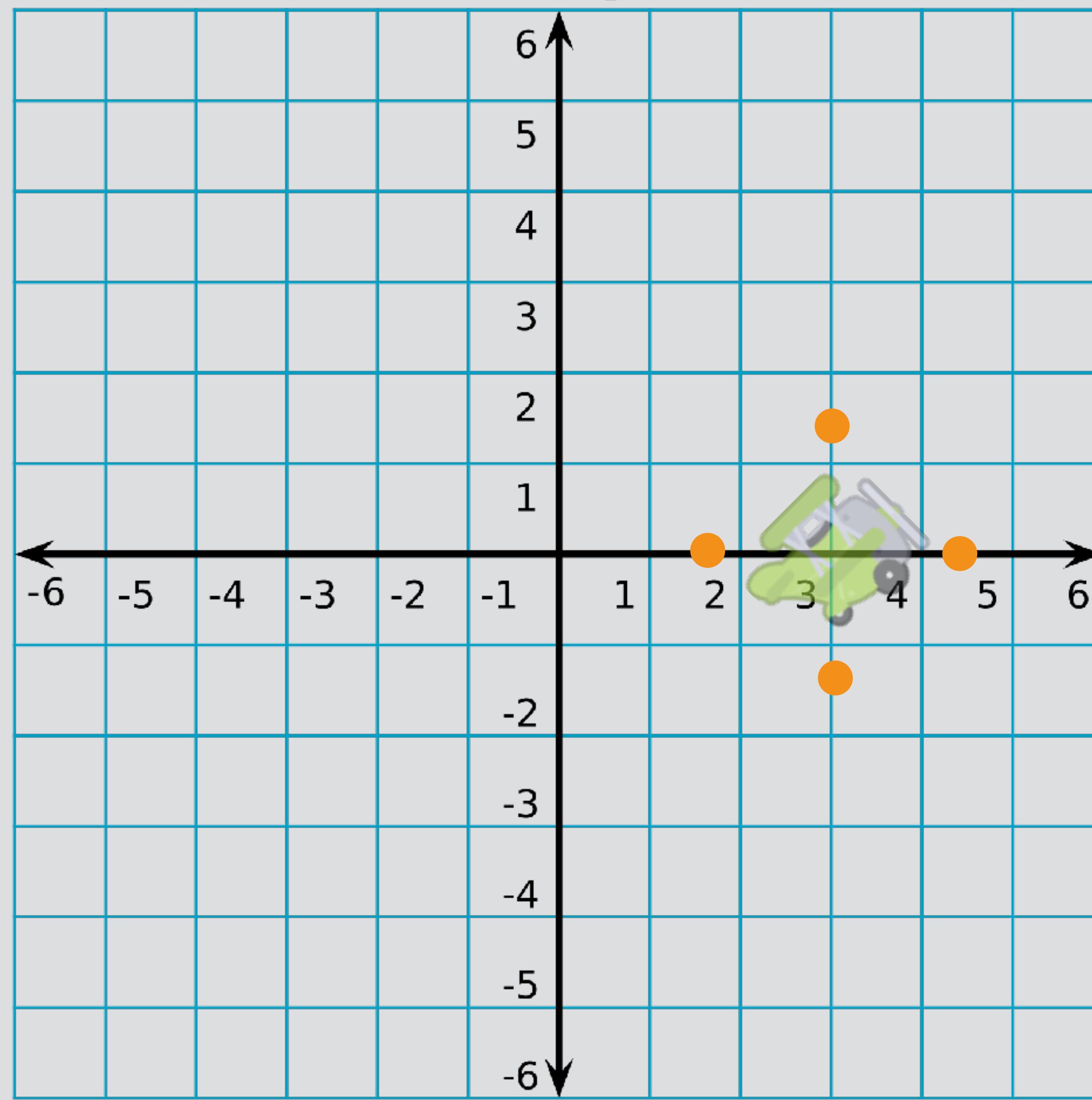
y-axis

x-axis



x-axis

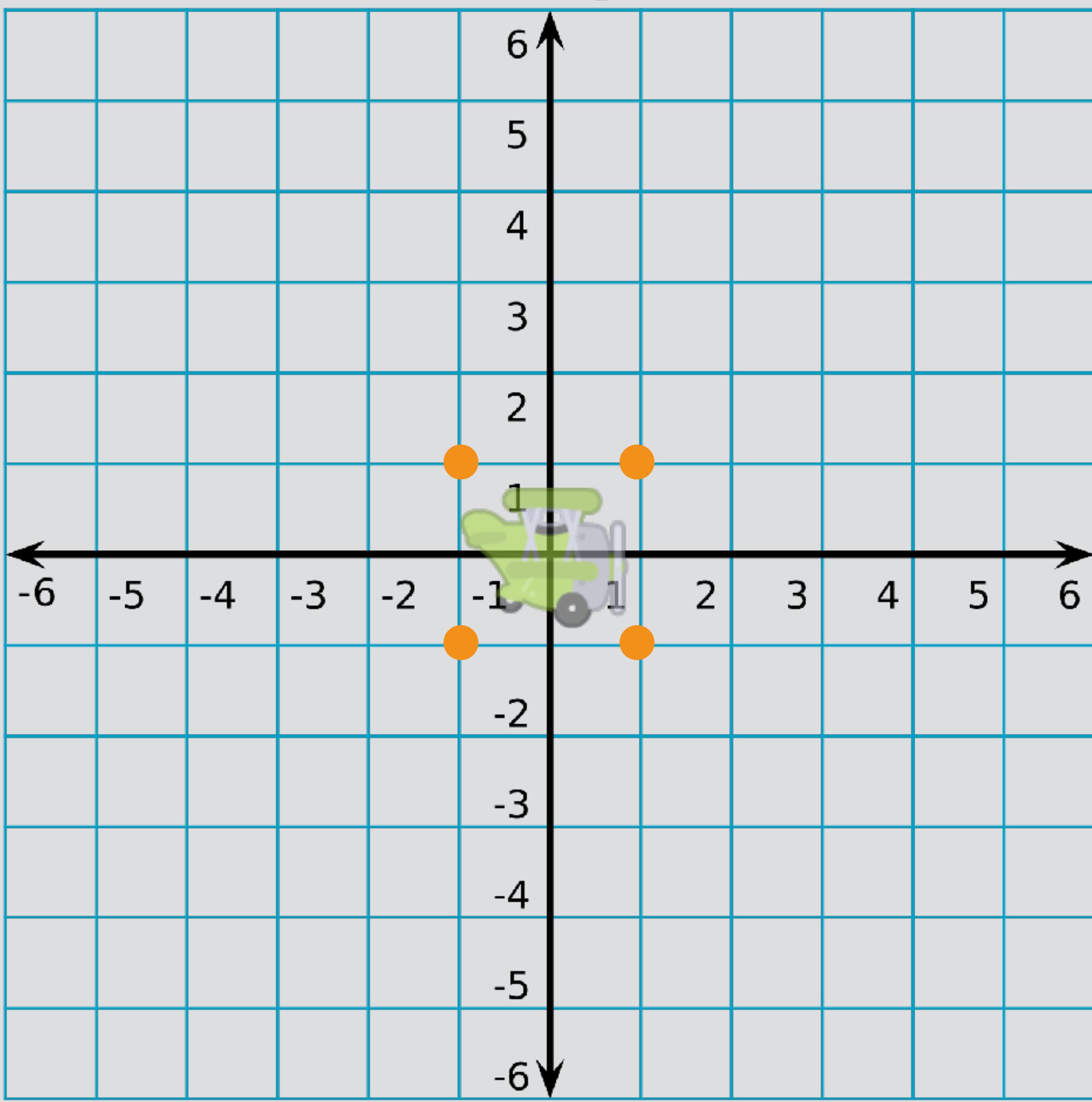
y-axis



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

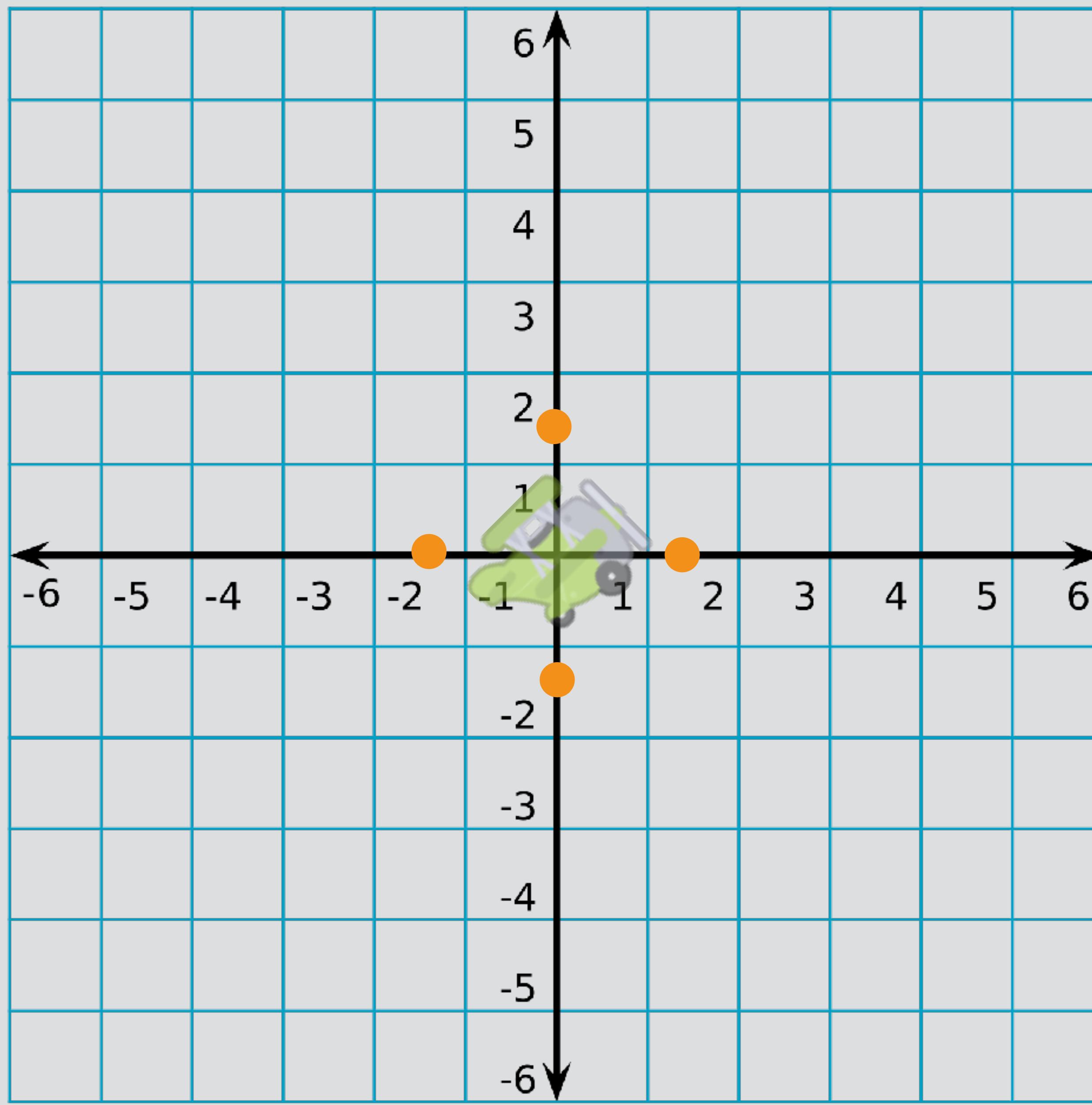
y-axis

x-axis



y-axis

x-axis



x-axis

y-axis

