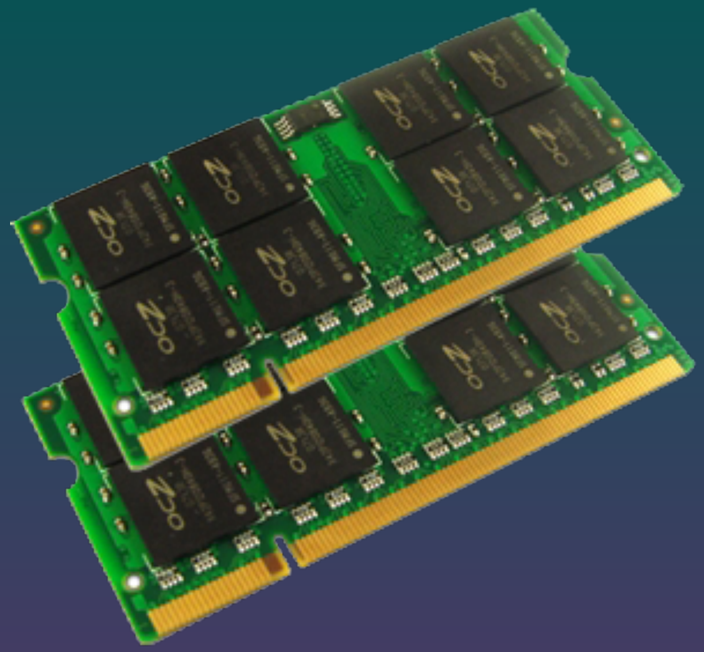
Graphics Foundations

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
```

0 1 2 3 4 5 6 7 8 9 …

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

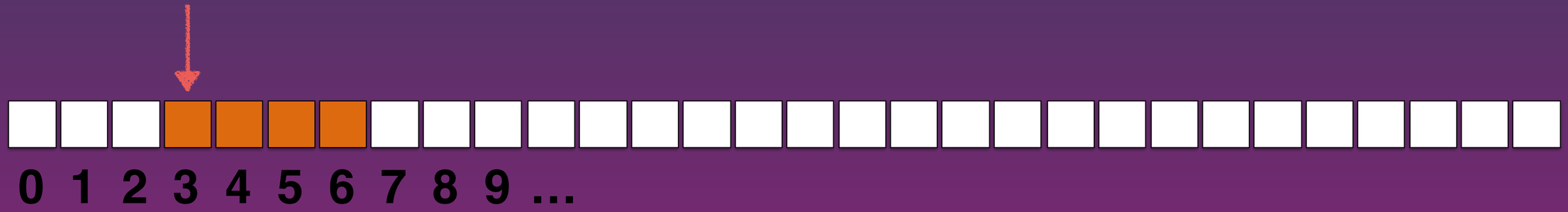
0 1 2 3 4 5 6 7 8 9 …

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

0 1 2 3 4 5 6 7 8 9 …

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

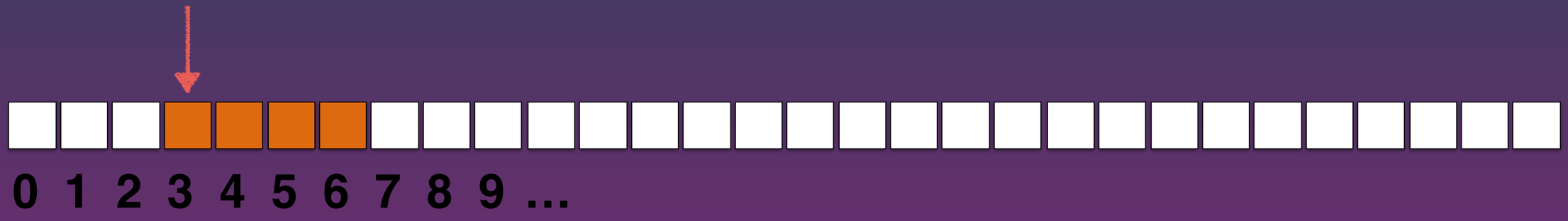

```
float *myVarPtr = &myVar;

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



```
float myVar = 16.0f;

float *myVarPtr = &myVar;

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

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

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

Dereference

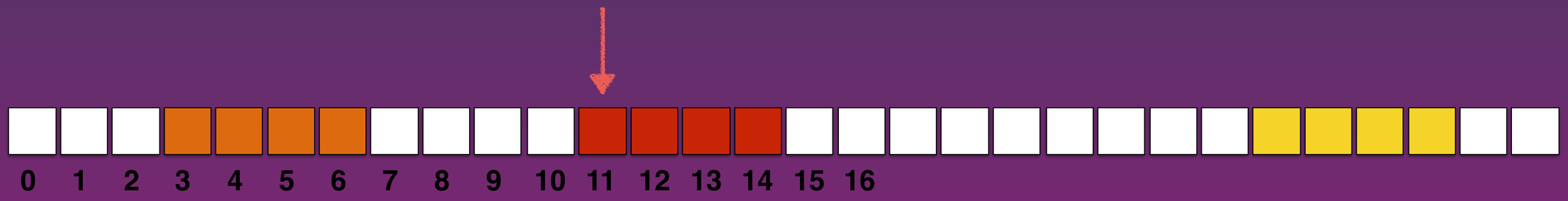
Reference

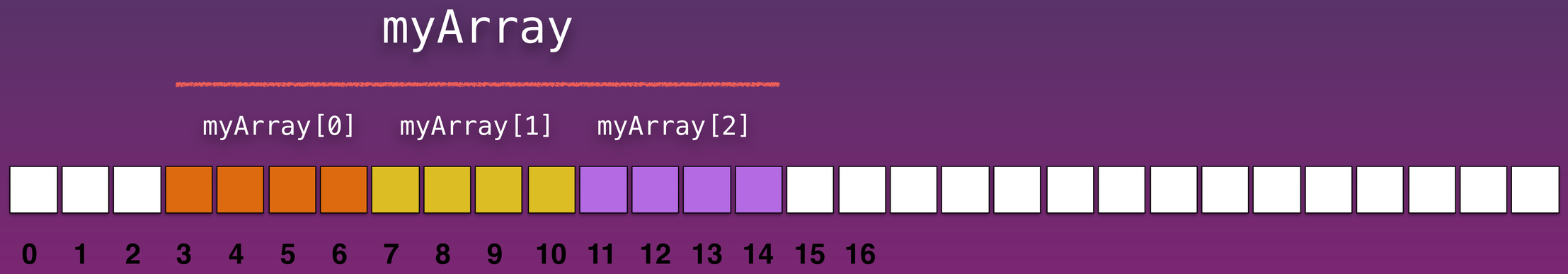


```
float myVar = 16.0f;

float *myVarPtr = &myVar;

cout << myVarPtr; // prints 3

cout << *myVarPtr; // prints 16

cout << &myVar; // prints 3

float **myVarPtrPtr = &myVarPtr;

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

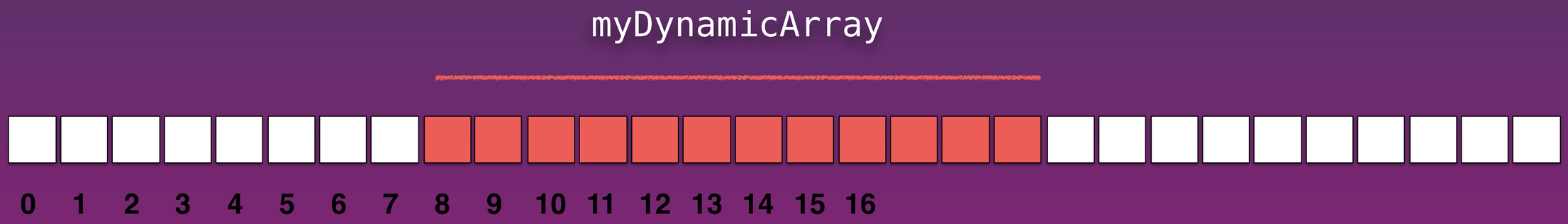
Dereference

Reference

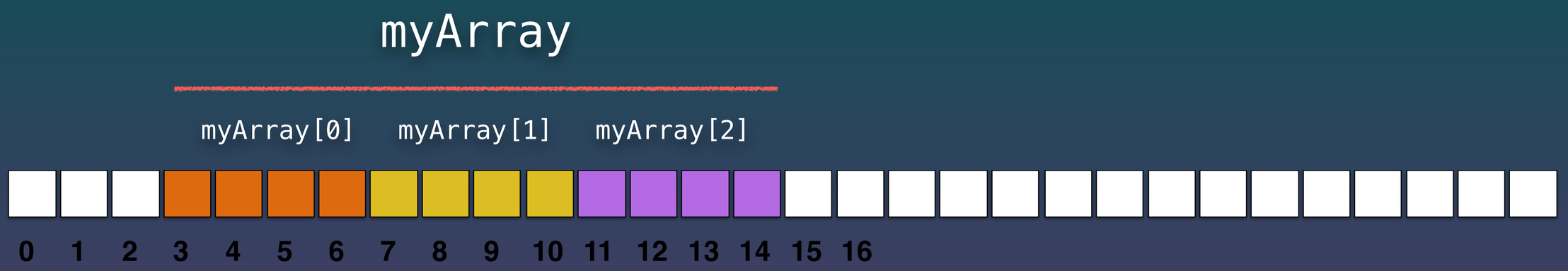


```float myArray[3] = {13.5f, 2.3f, 5.4f};

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```


```

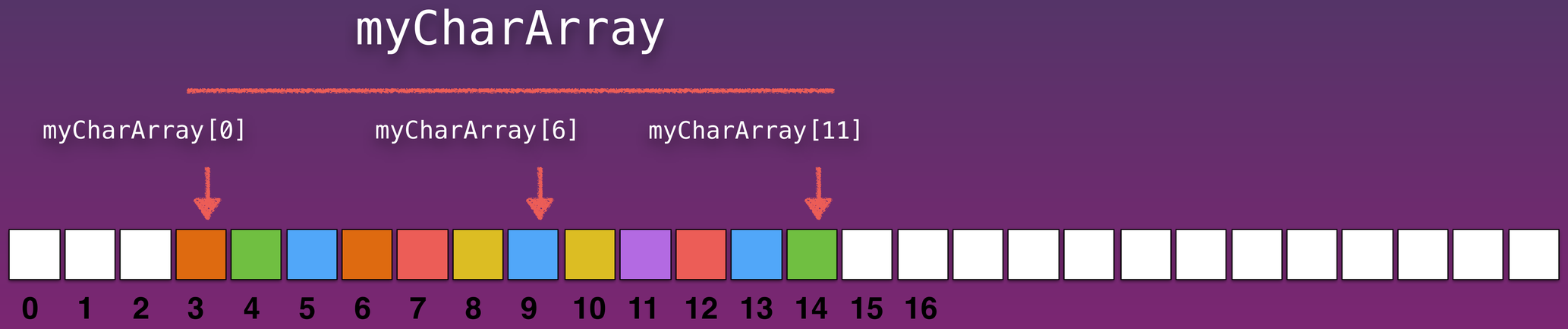
```
float *myDynamicArray;

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;
```



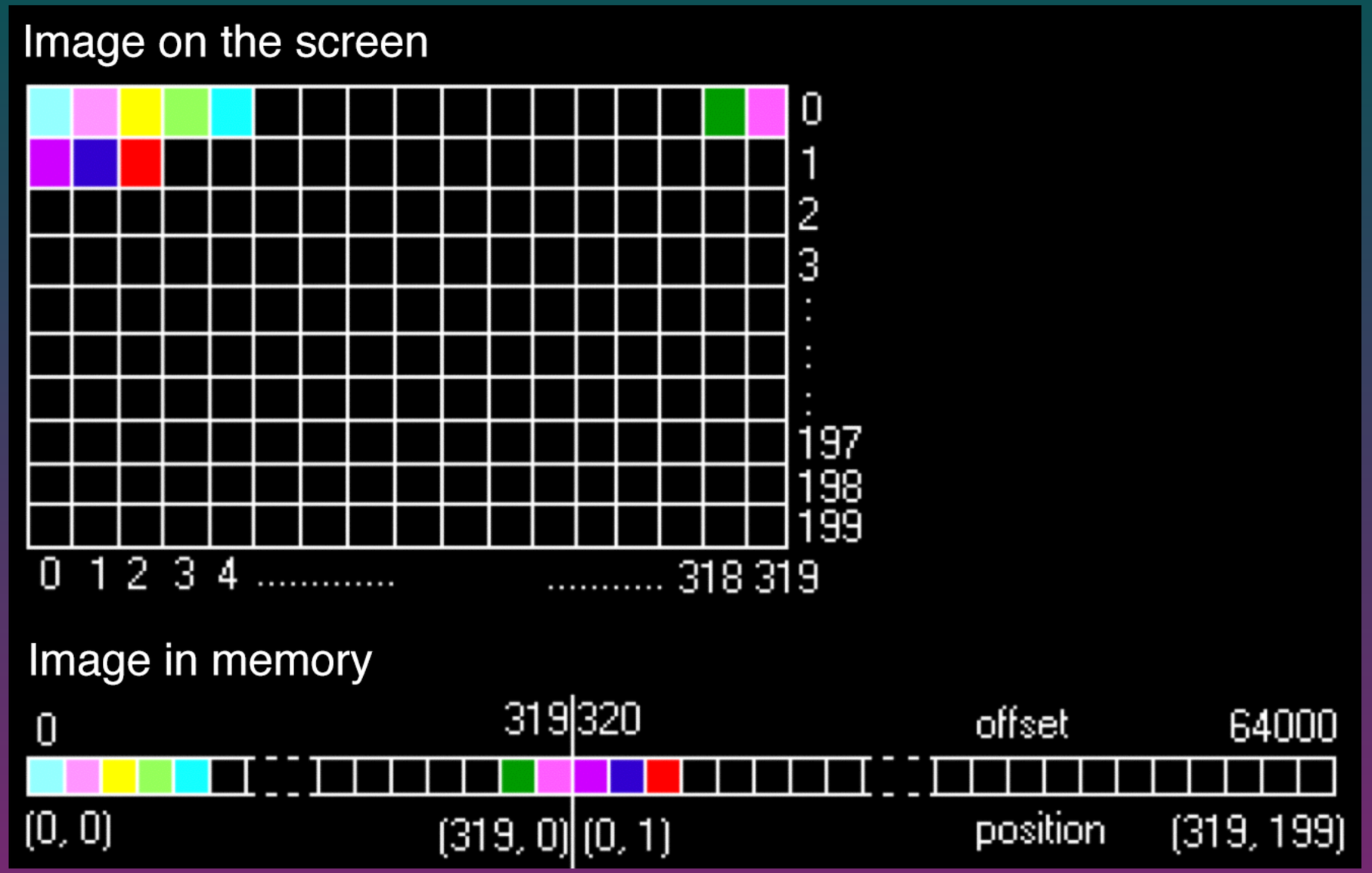
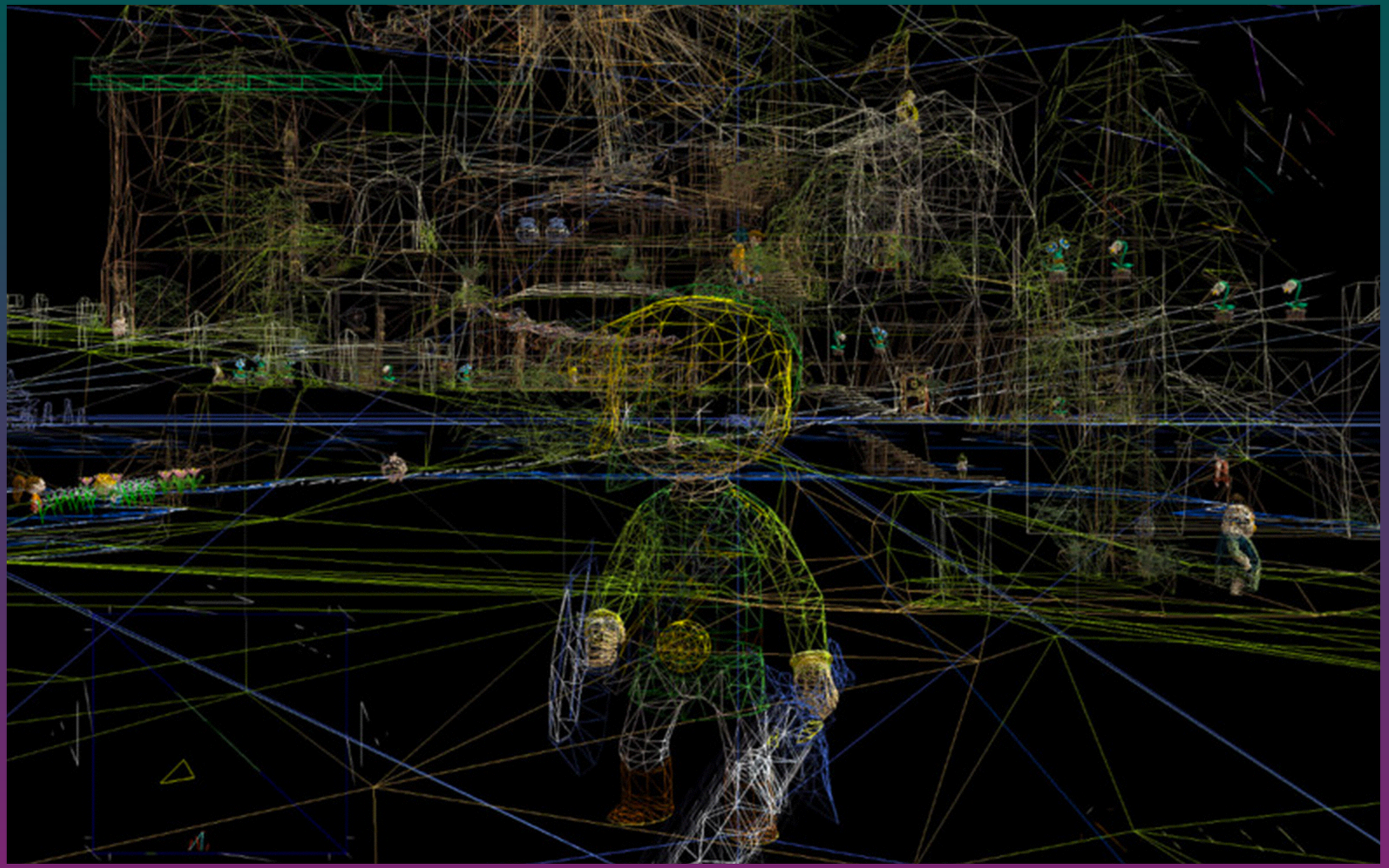
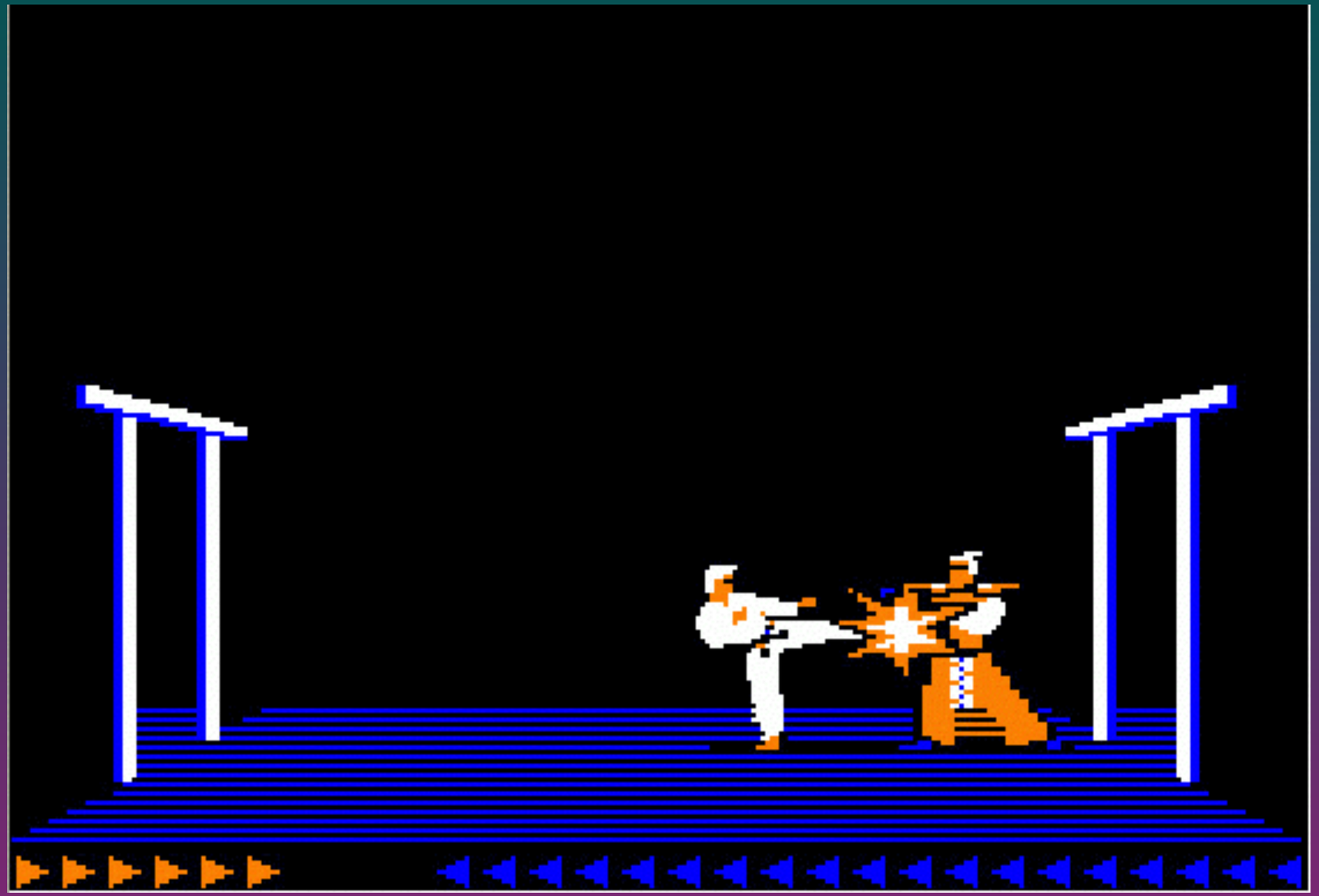


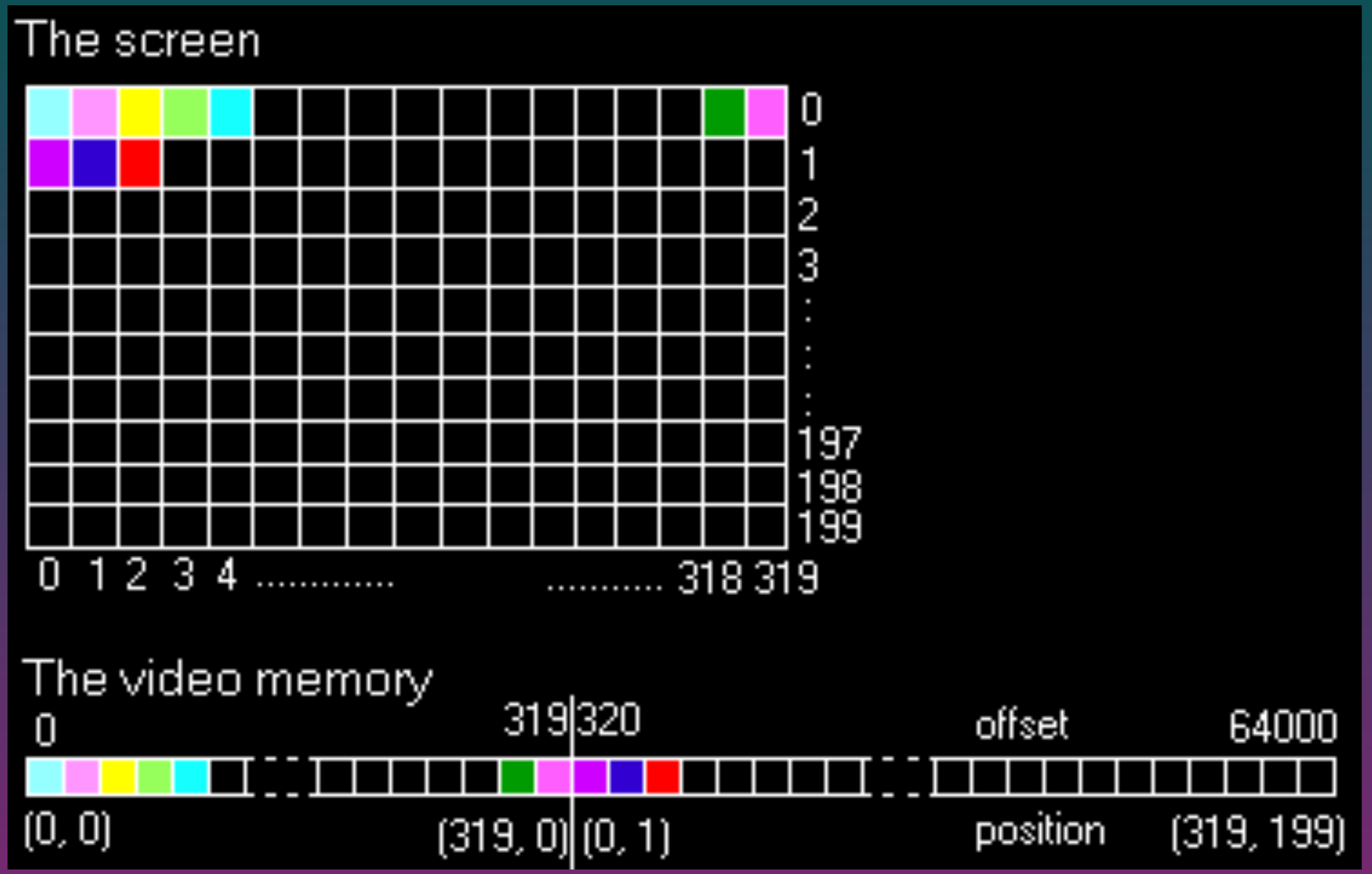
Image on the screen
0
1
2
3
197
198
199
0 1 2 3 4 .... 318 319
Image in memory
0
319
320
offset
64000
(0, 0)
(319, 0)
(0, 1)
position
(319, 199)

Graphics in games


Put Away
72
FREE
4168



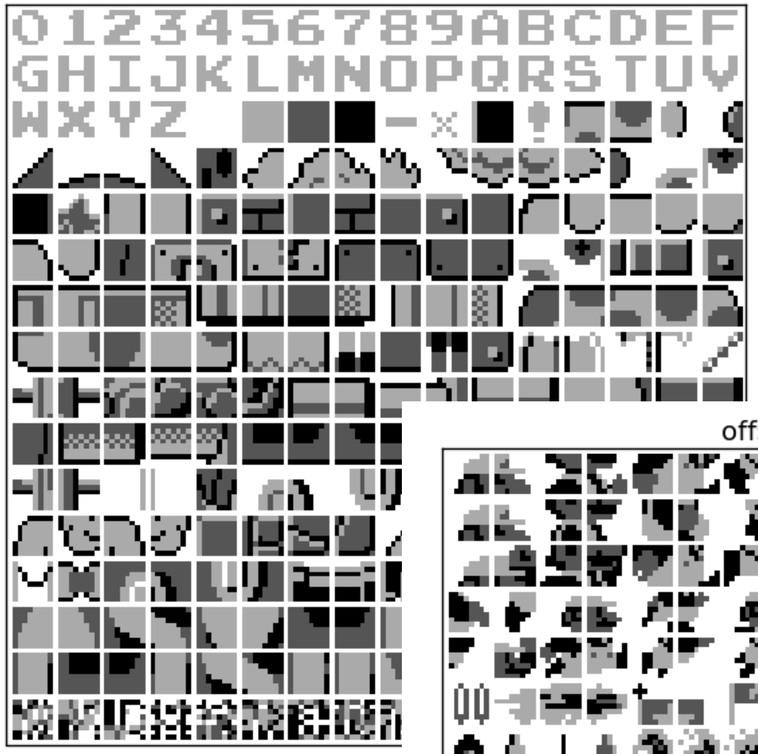



The screen
0
1
2
3
197
198
199
0 1 2 3 4 .... 318 319
The video memory
0
319
320
offset
64000
(0, 0)
(319, 0)
(0, 1)
position
(319, 199)


SUPER
PAC-MAN

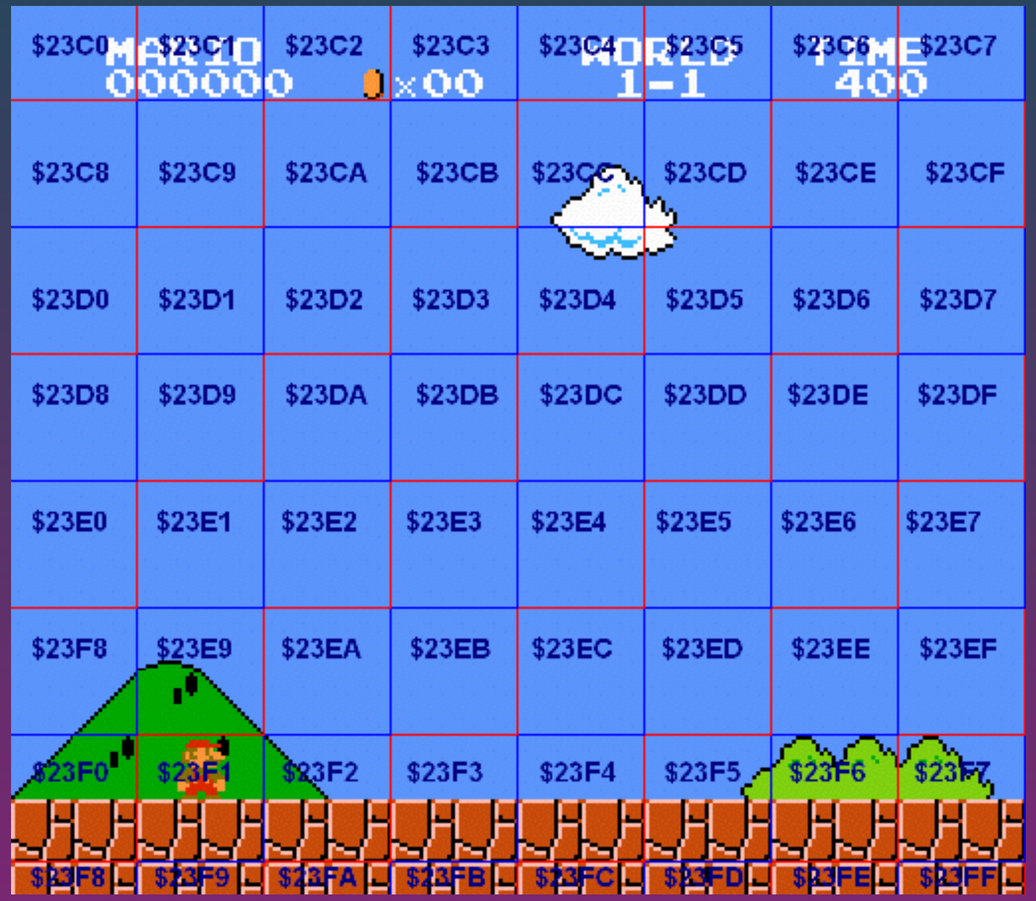

MARIO
000000
×00
WORLD
1-1
TIME
SUPER
MARIO BROS.
©1985 NINTENDO
1 PLAYER GAME
2 PLAYER GAME
TOP- 000000

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




KITTY
000200
x01
WORLD
1-1
TIME
368


Load New Commander (Y/N)?

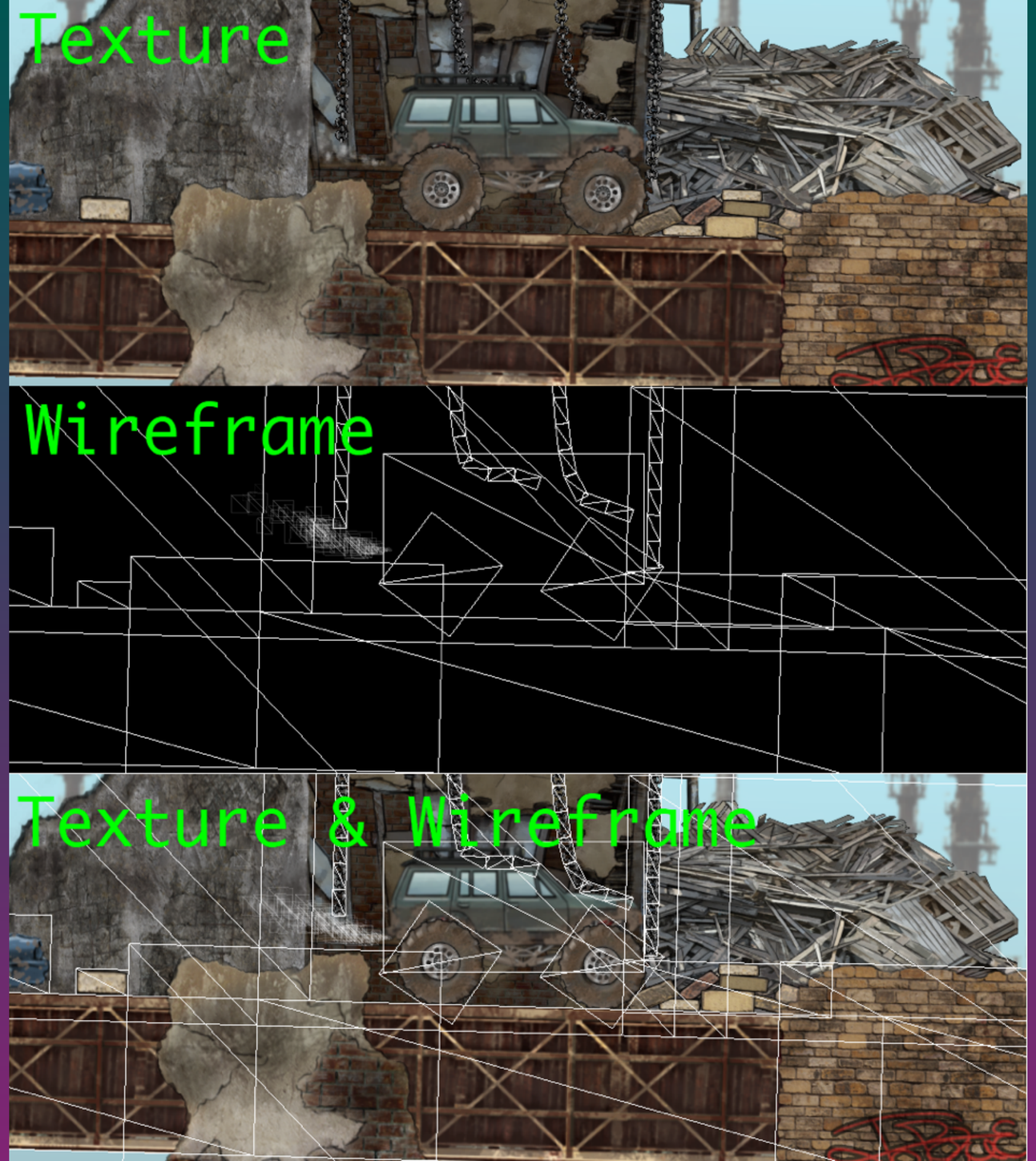

SPD: 0
VSI: 01.78
THR:90%
HDG:0
ALT:83
ZOOM:2


93
75
13
0
78
91
13


20
0
0
11
108

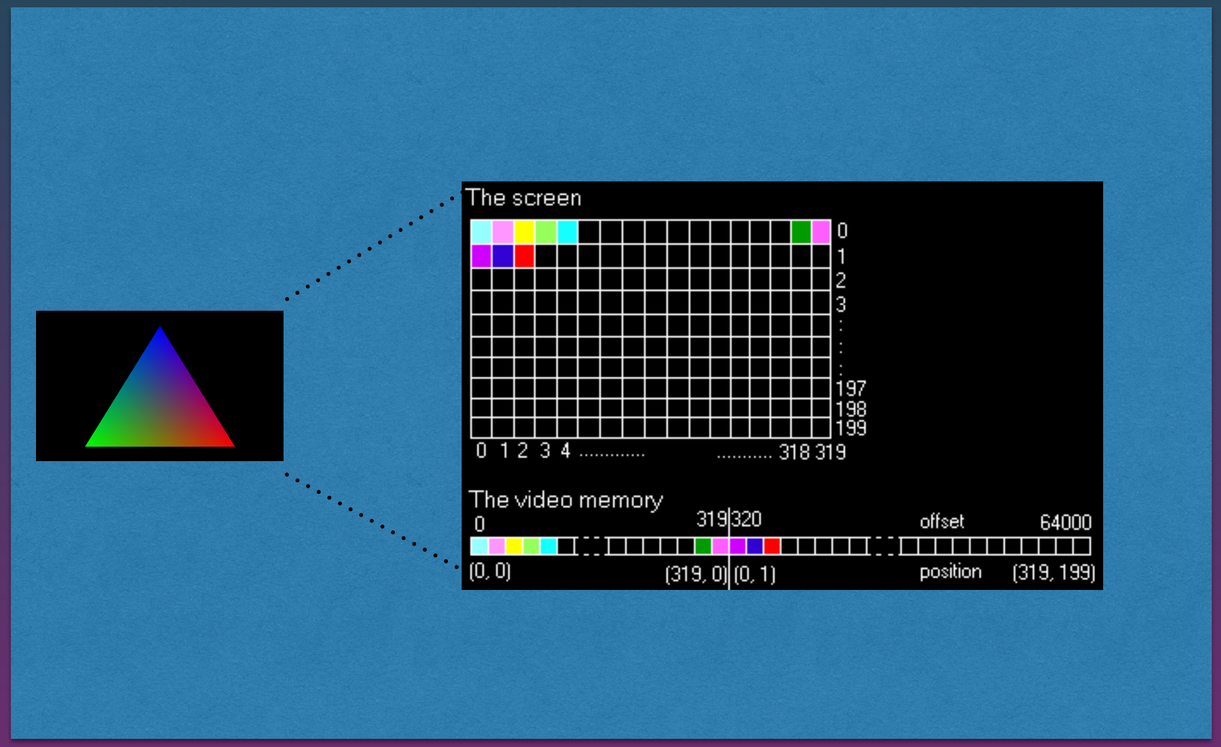

12MB
3D
BLASTER
Hardware XTRA AWARD
PC-Games 05/98
Hardware XTRA AWARD
PC-Action 05/98
PC Referenz
PC-Player 05/98
Voodoo 2
FASZINIERENDE SPIELE
4
FASZINIERENDE SPIELE
INCOMING
actua SOCCER 2
ULTIMATE RACE PRO
3Dfx
Get the Magic of Speed!
Basiert auf dem neuen Voodoo2 Graphics™ Chipsatz von 3Dfx Interactive™
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


PARK
ONE W

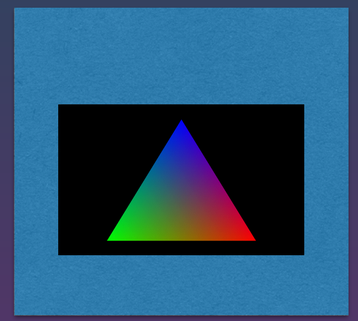

Texture
Wireframe
Texture & Wireframe

Software Rendering

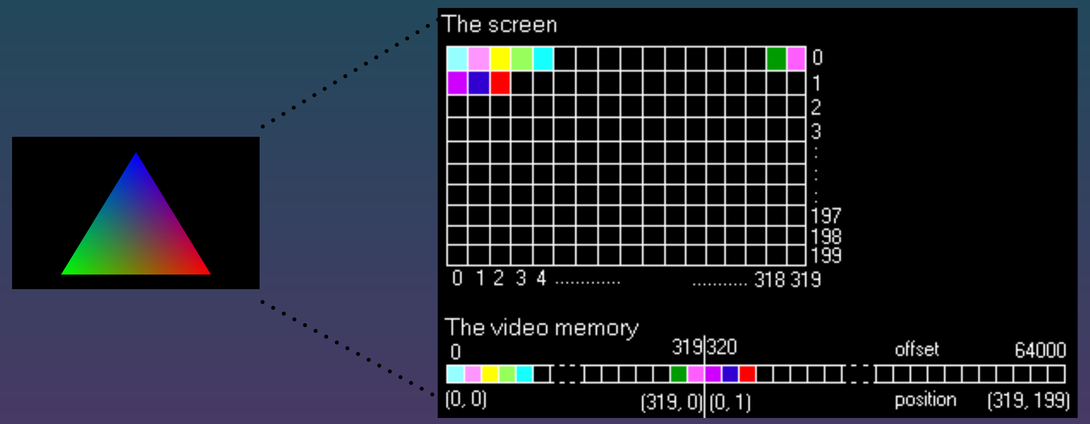
RAM



Video card



Software Rendering

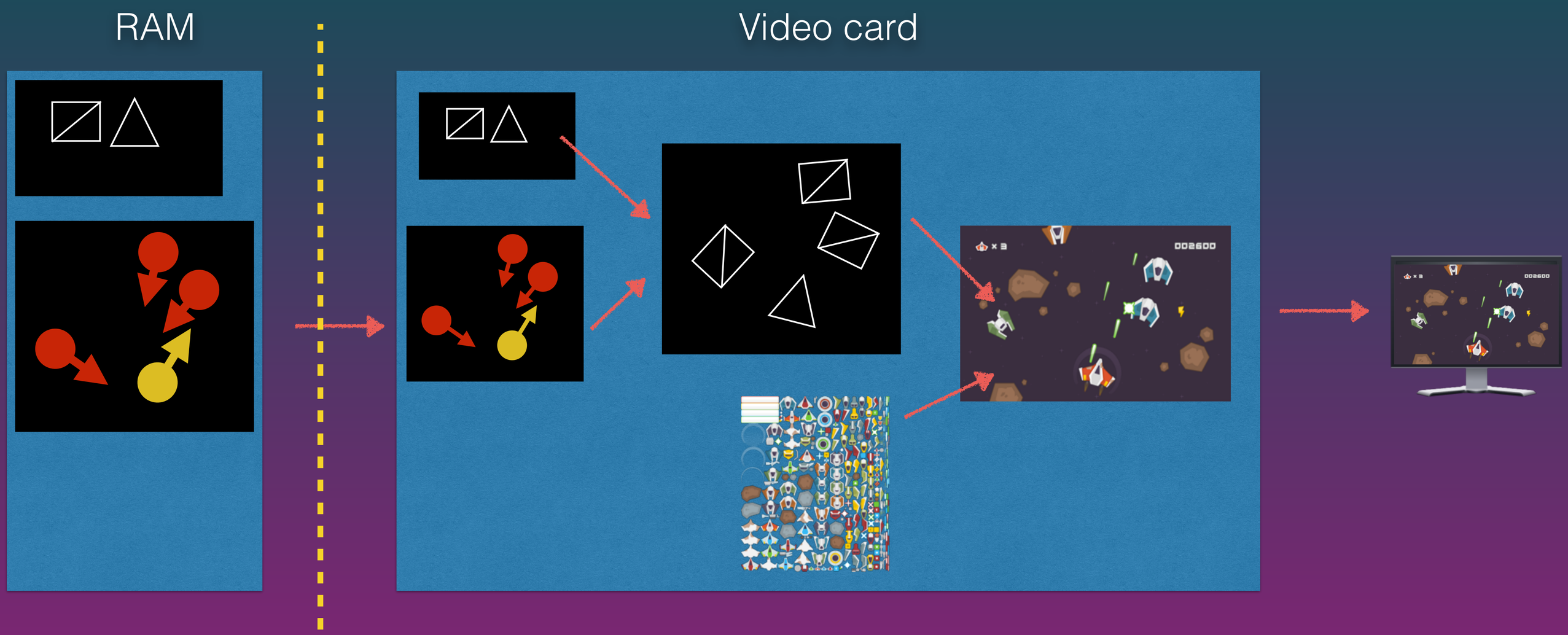


2560 * 1600 = 4,096,000 pixels

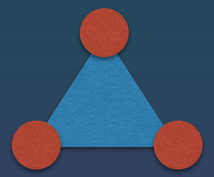
~12 MB == 4 * sizeof(War and Peace)

60 times a second

Hardware Rendering



The rendering pipeline.



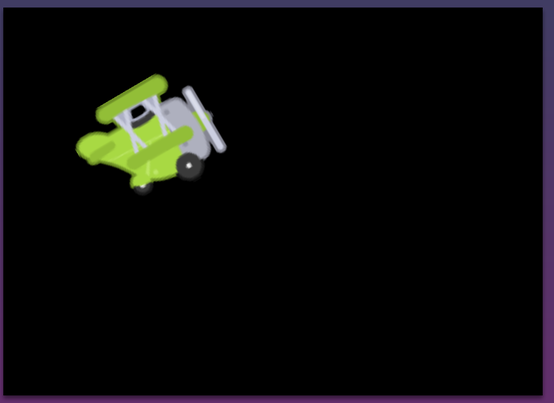
MODELVIEW MATRIX

PROJECTION MATRIX

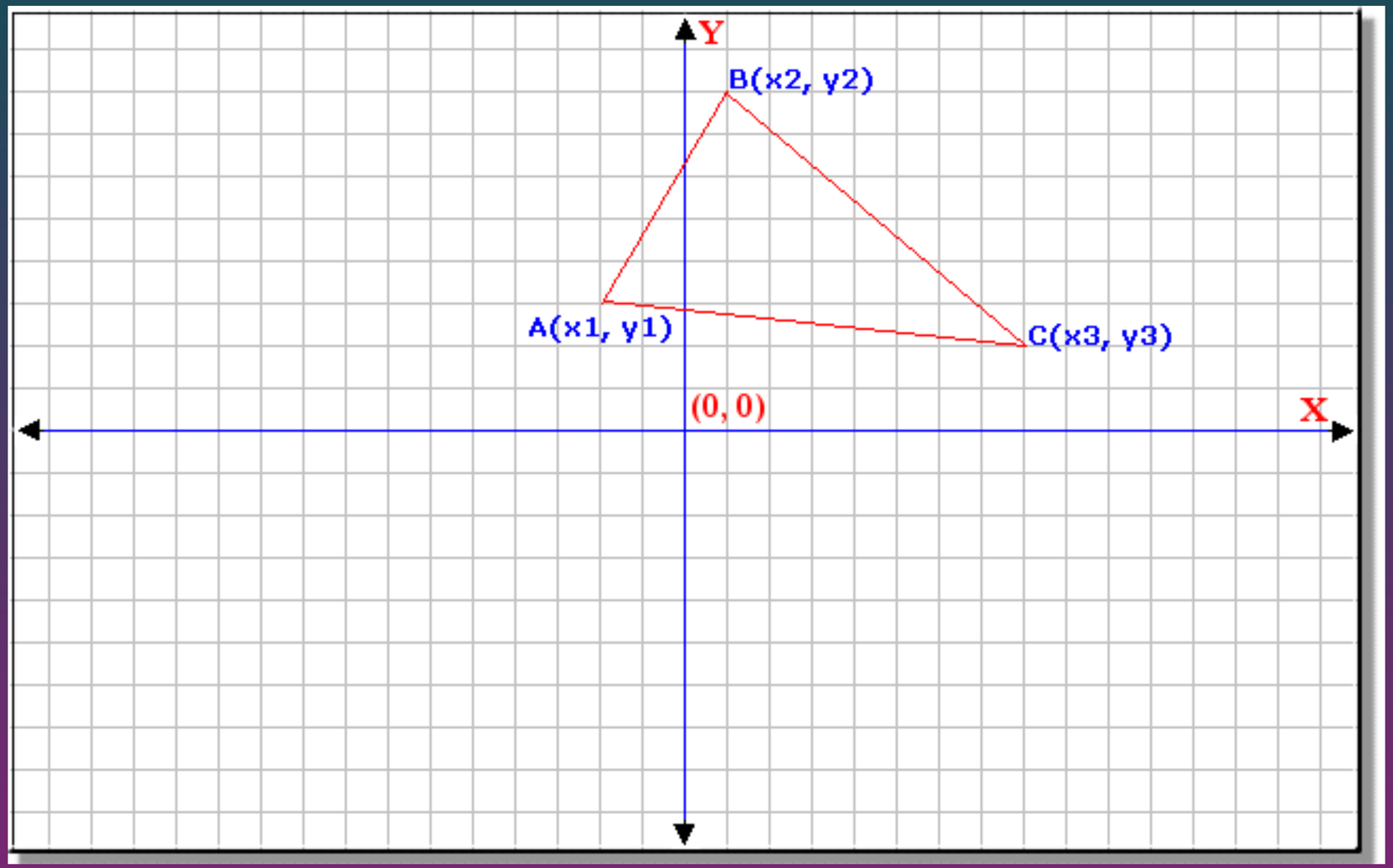
VERTEX SHADER

VIEWPORT MAPPING

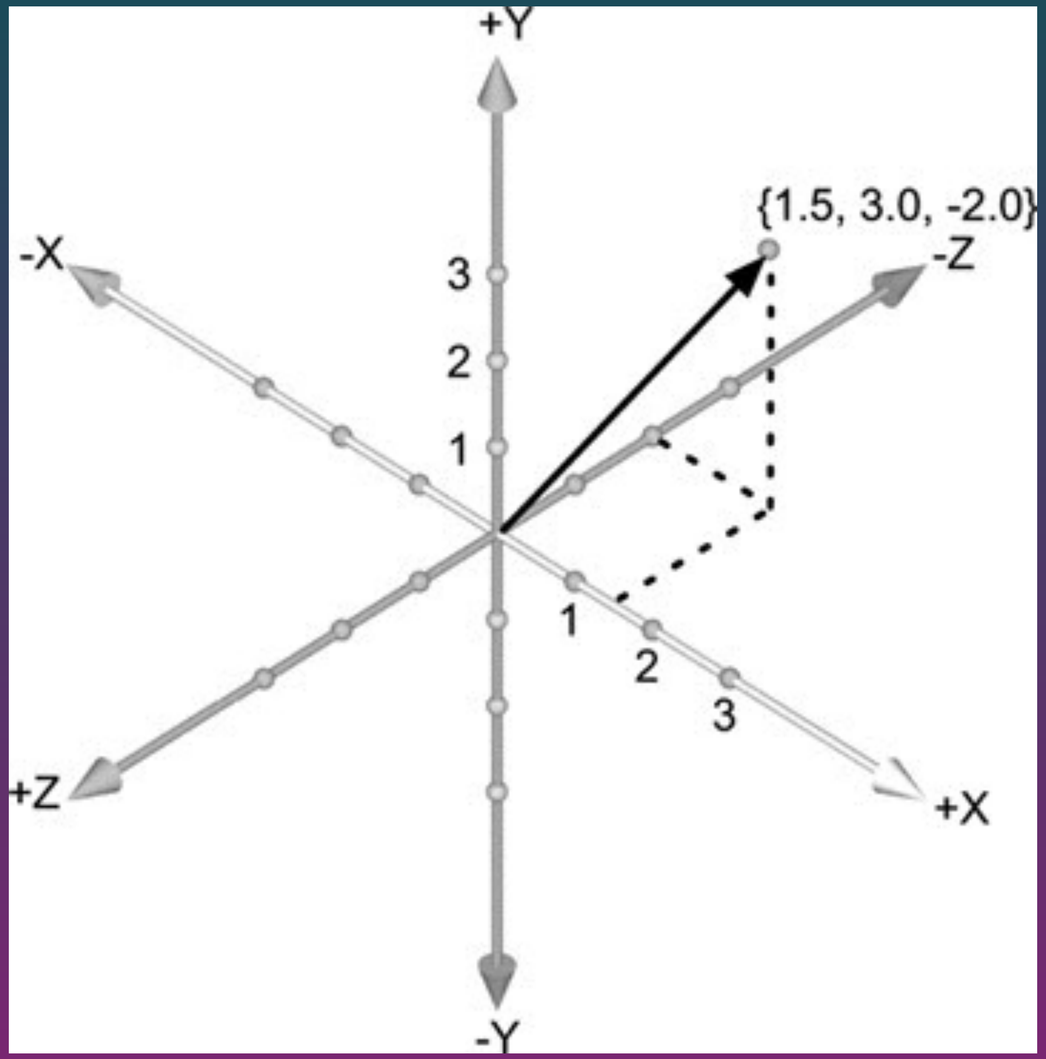
FRAGMENT SHADER

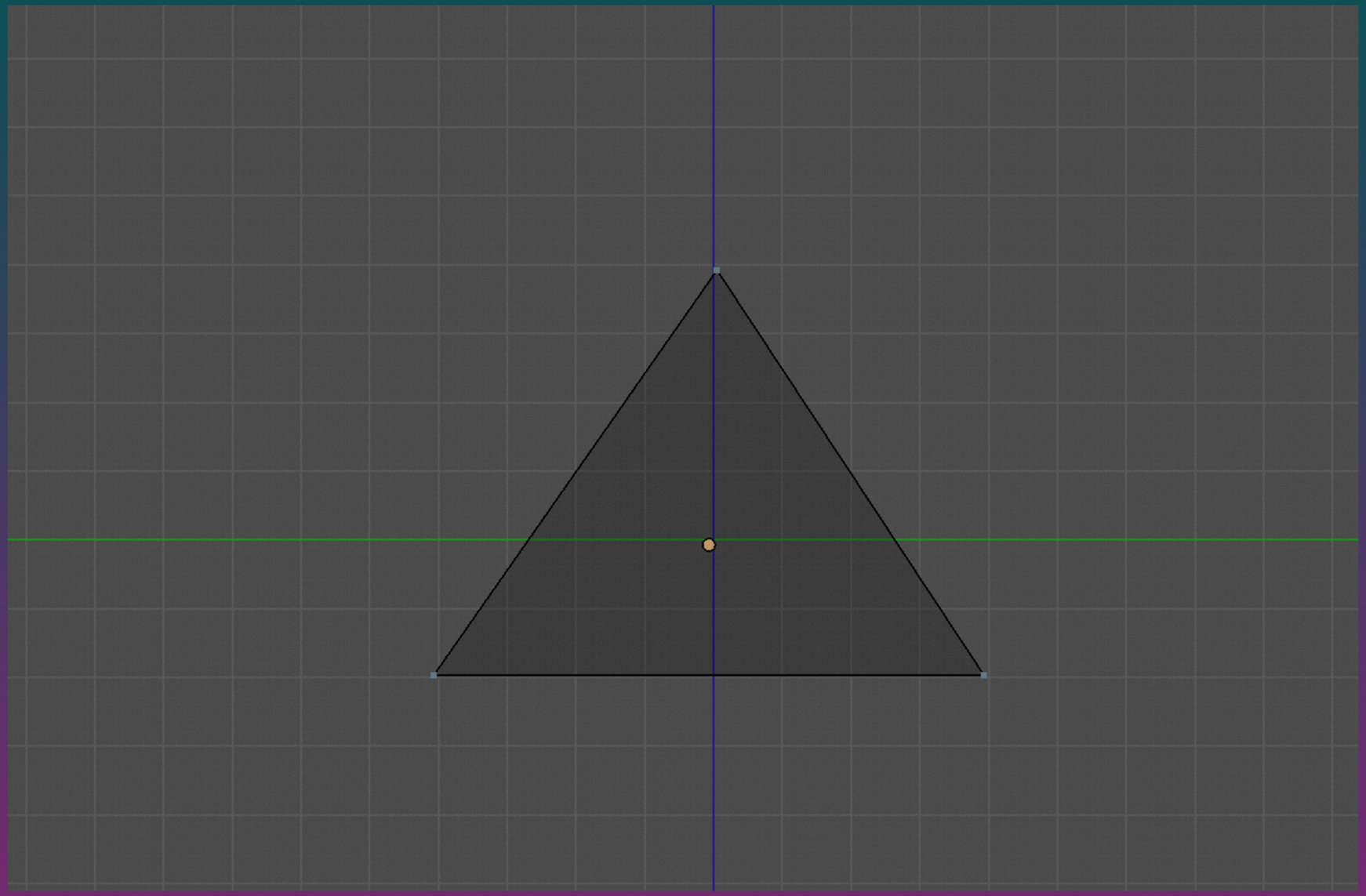


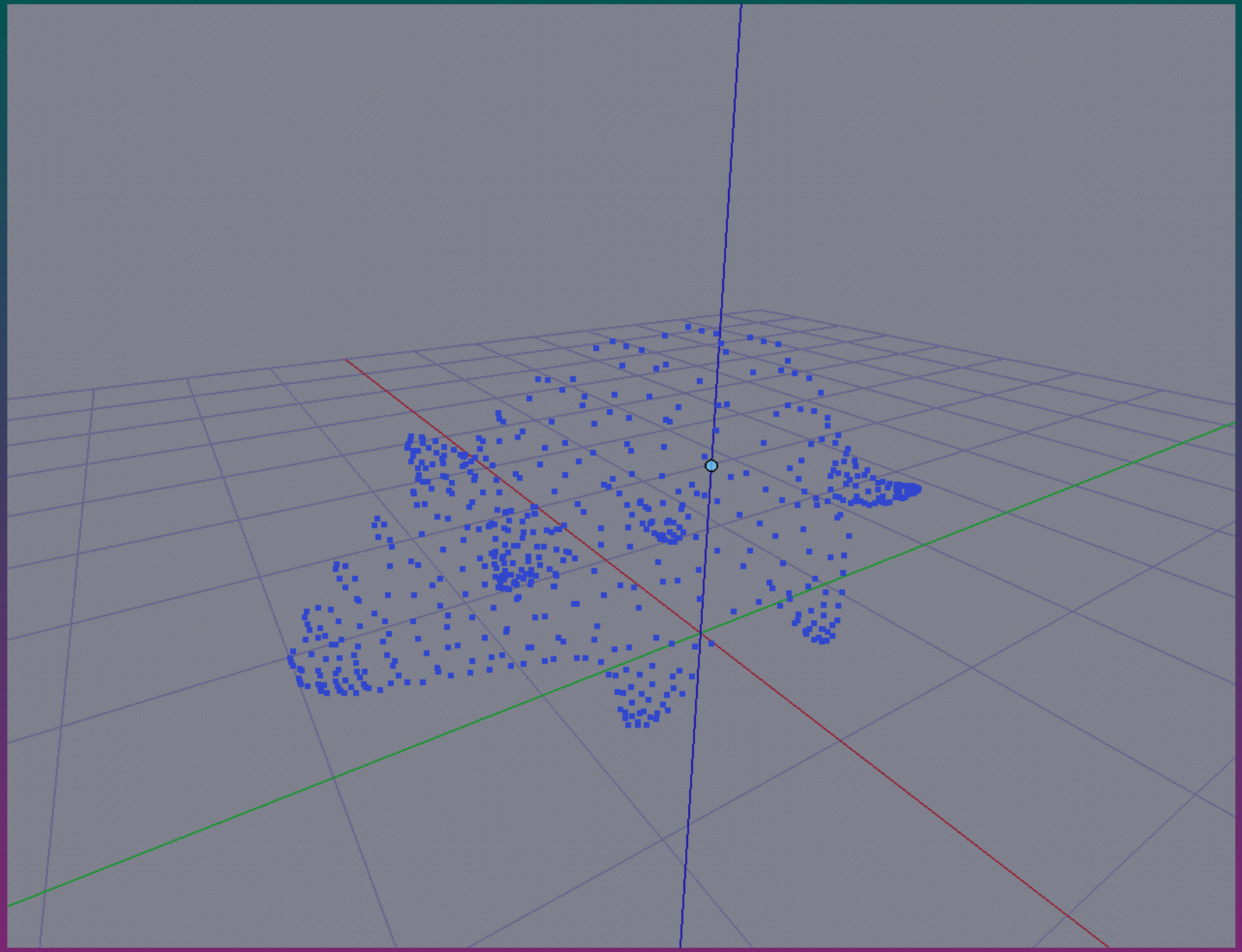
Vertex data

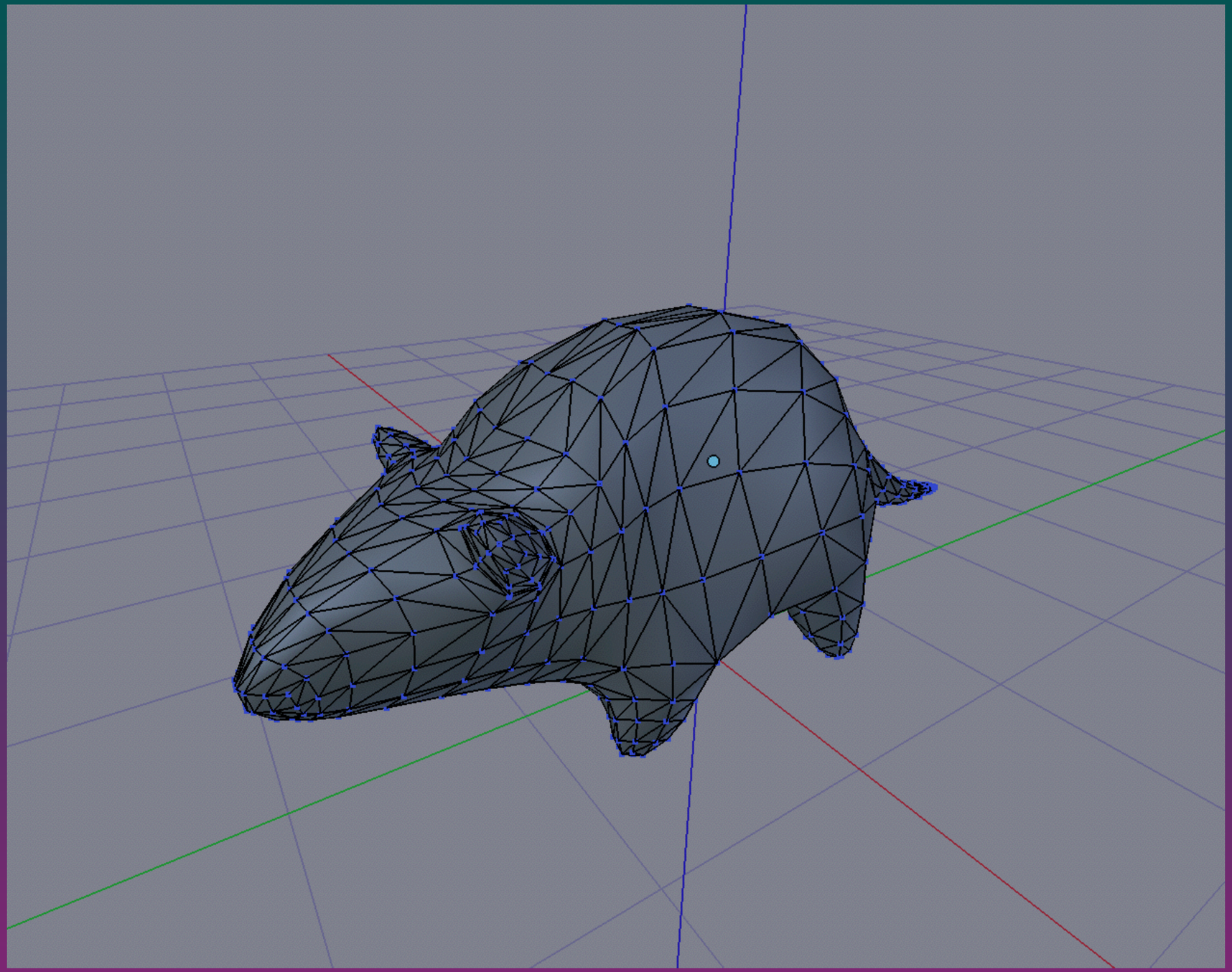

Y
B(x2, y2)
A(x1, y1)
C(x3, y3)
(0, 0)
X

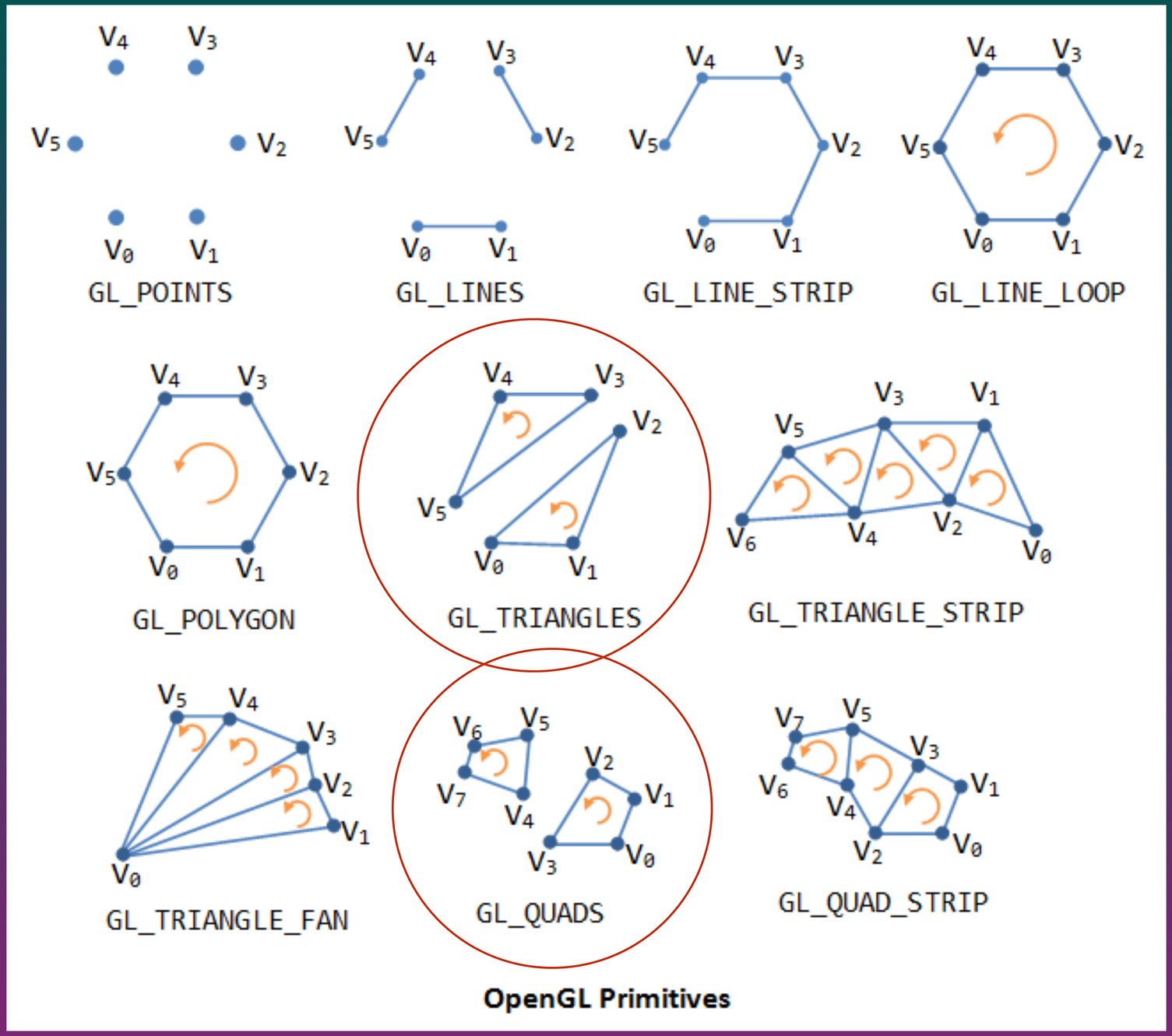
Vertices



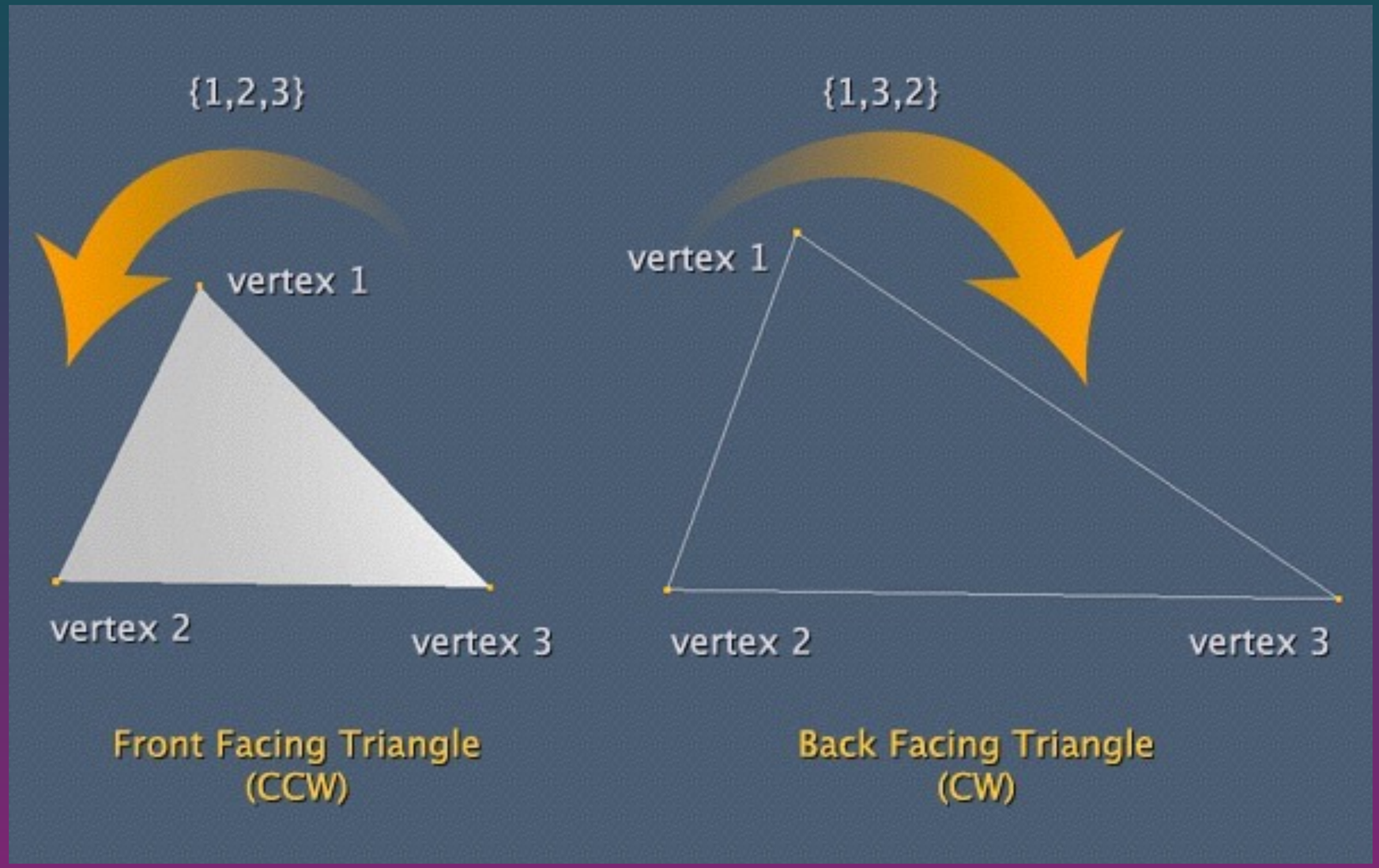


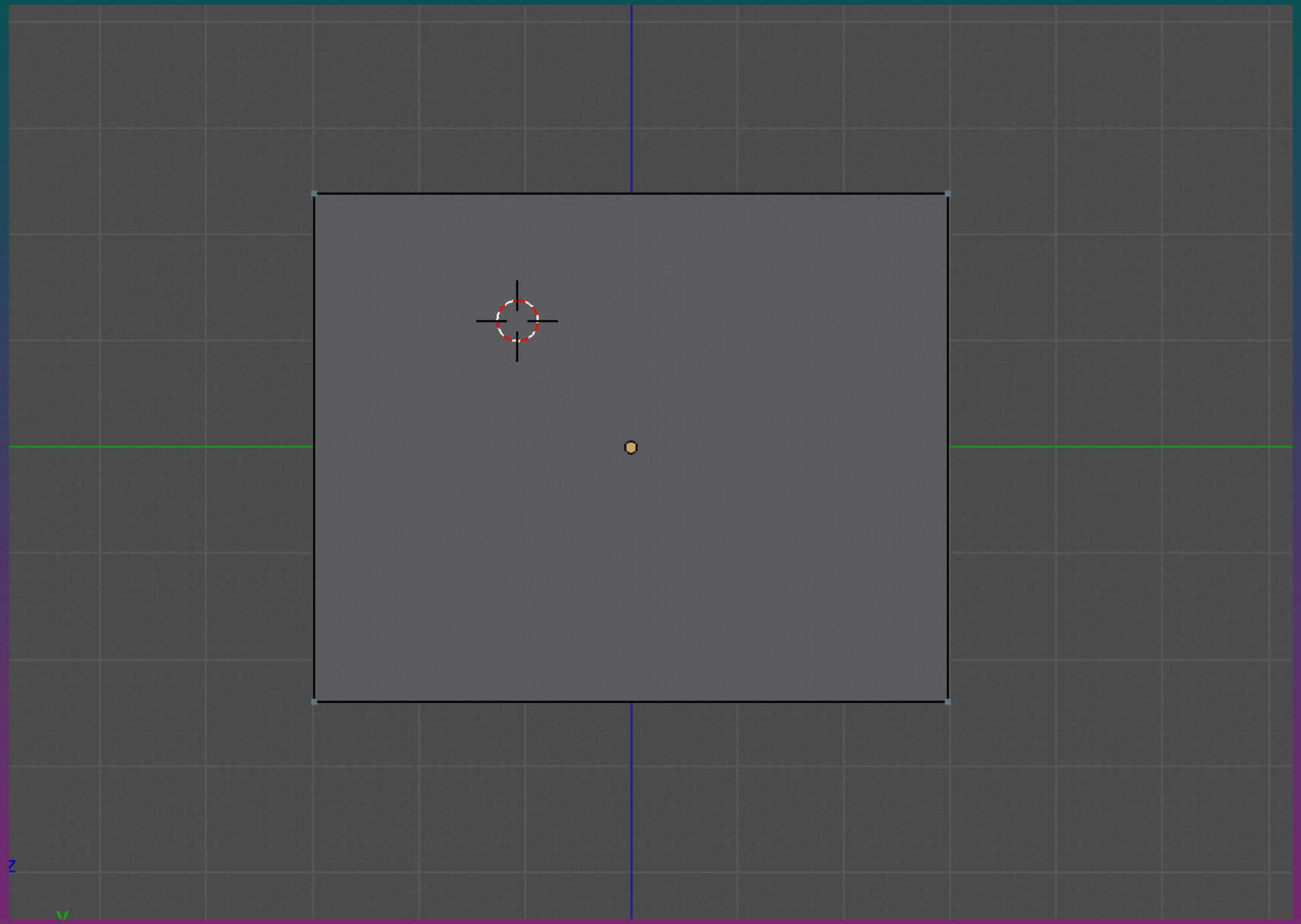


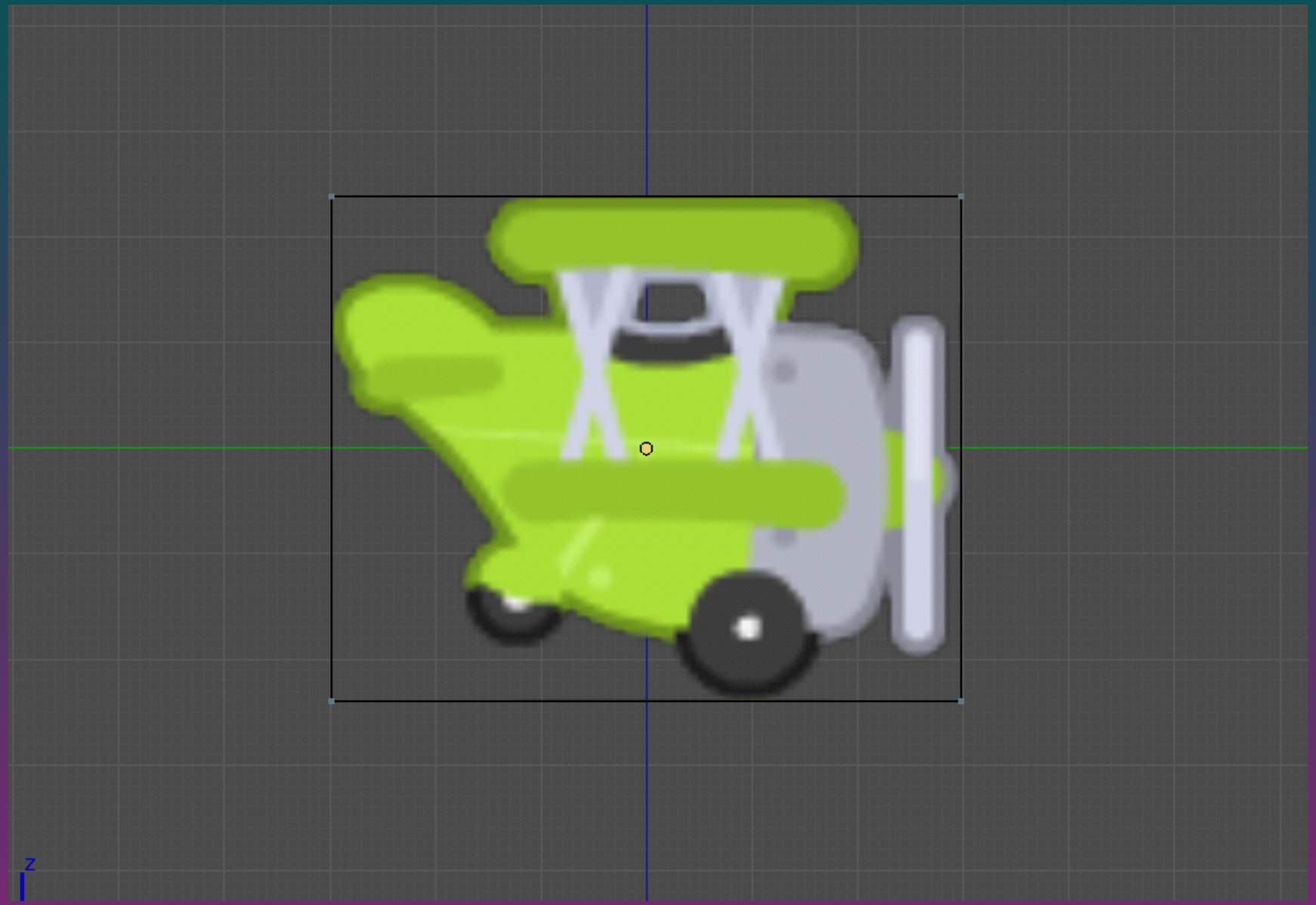



V4 V3
V5 V2
V0 V1
GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_POLYGON
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP
OpenGL Primitives

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




Z
Y


z

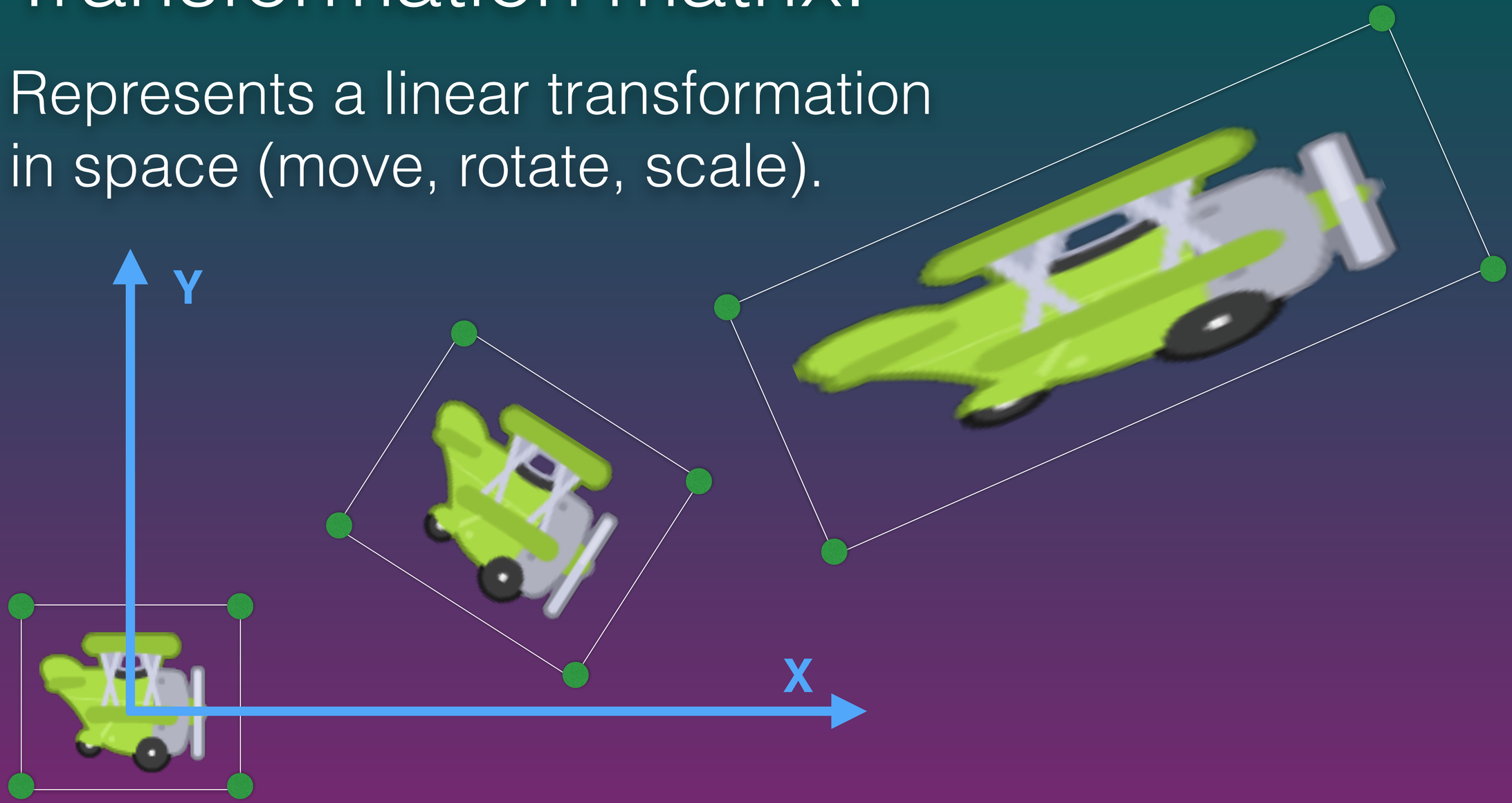
Modelview matrix.

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.

Represents a linear transformation in space (move, rotate, scale).



Vertex position * transformation matrix = new vertex position

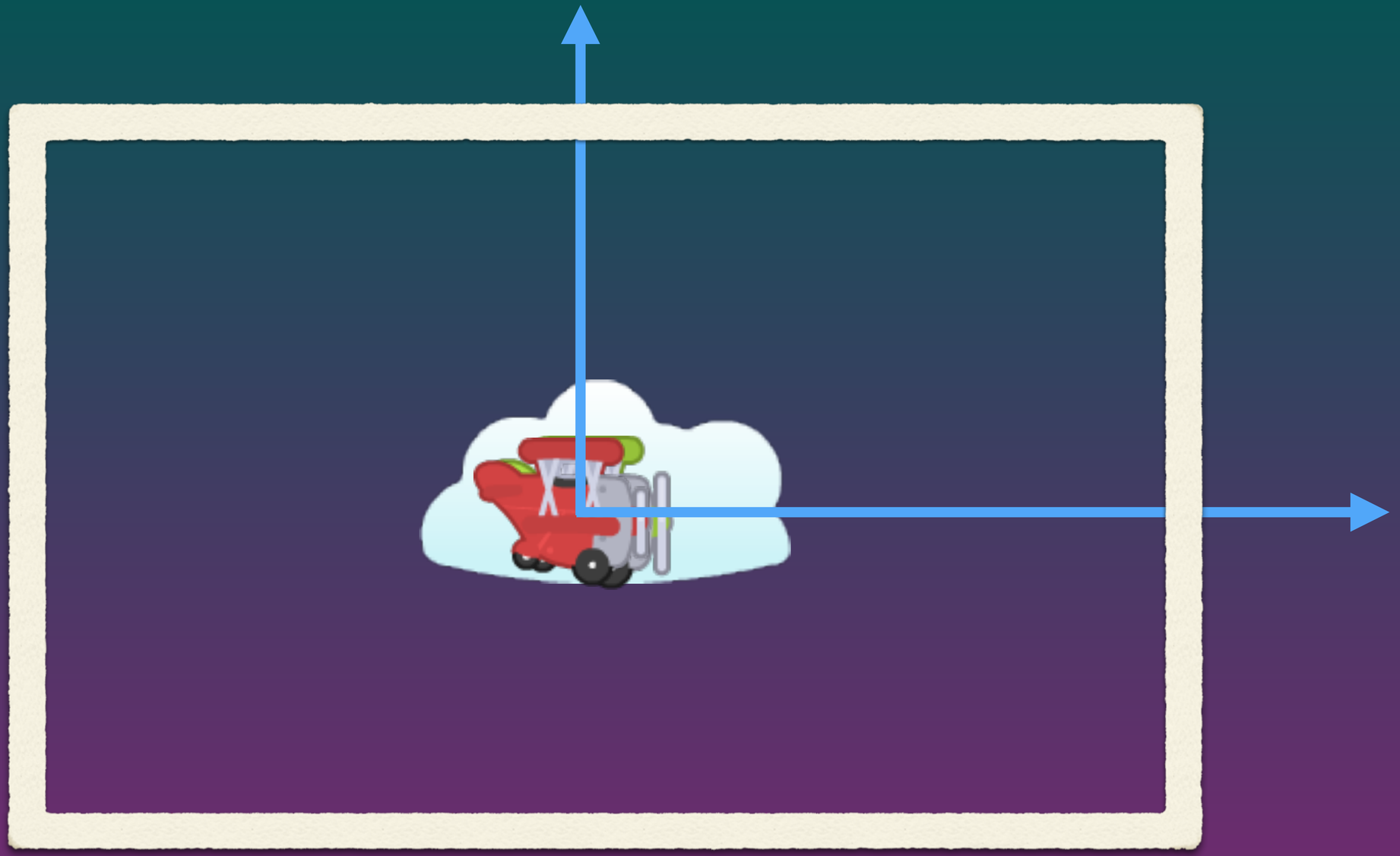
Modelview matrix

=

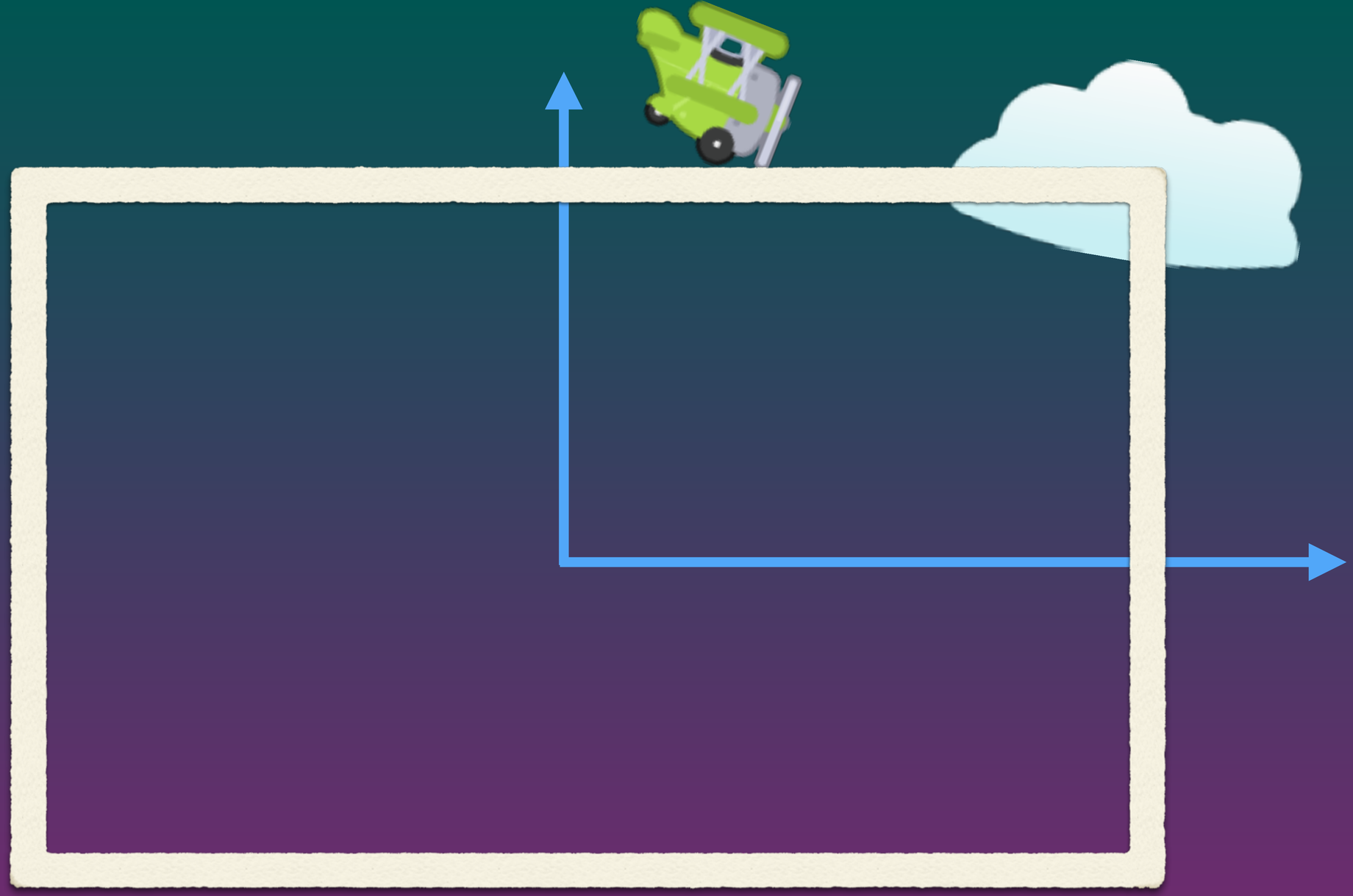
Model transformation matrix

*

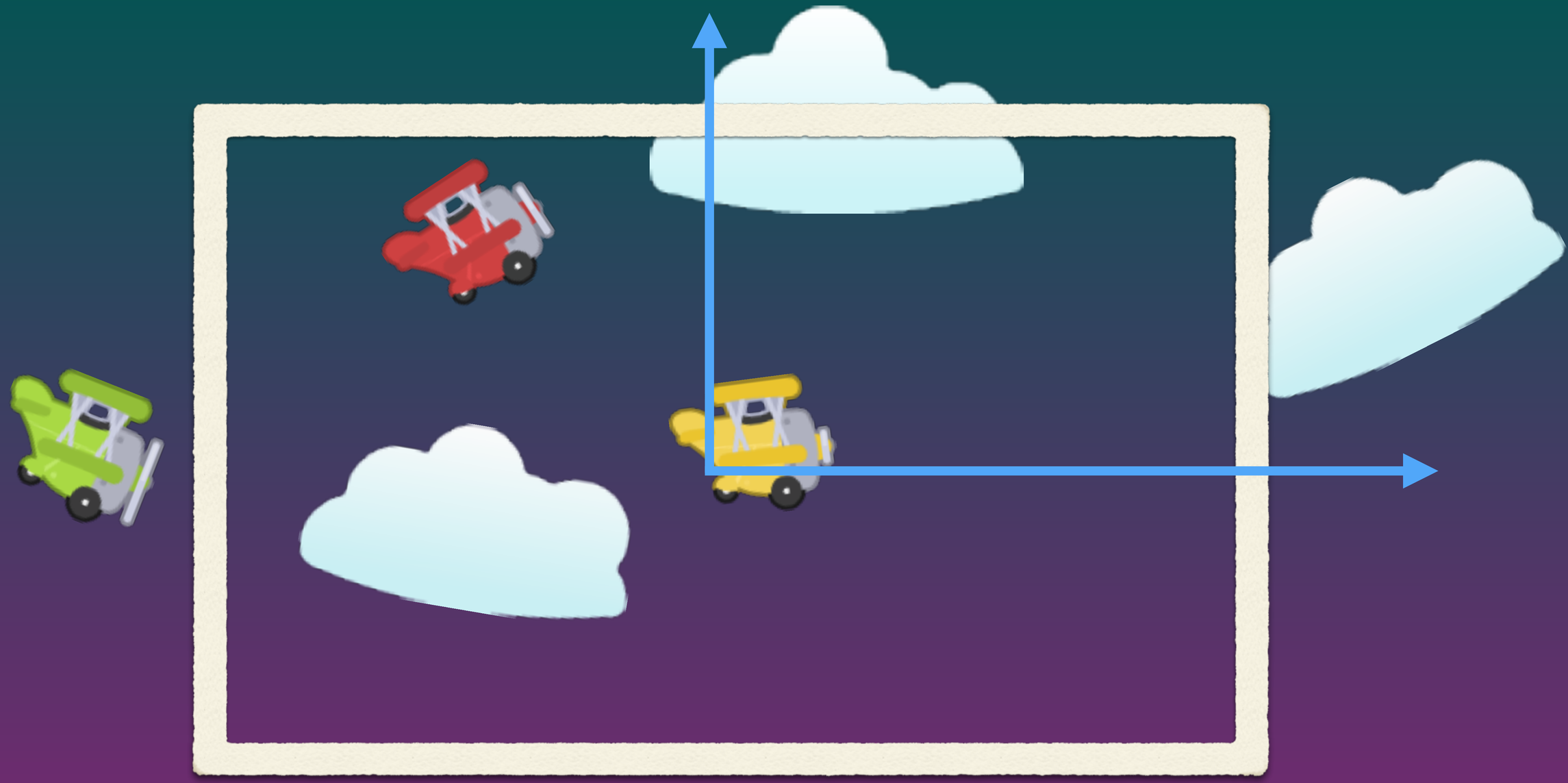
View transformation matrix.



Without model or view matrices :(



Without view matrix

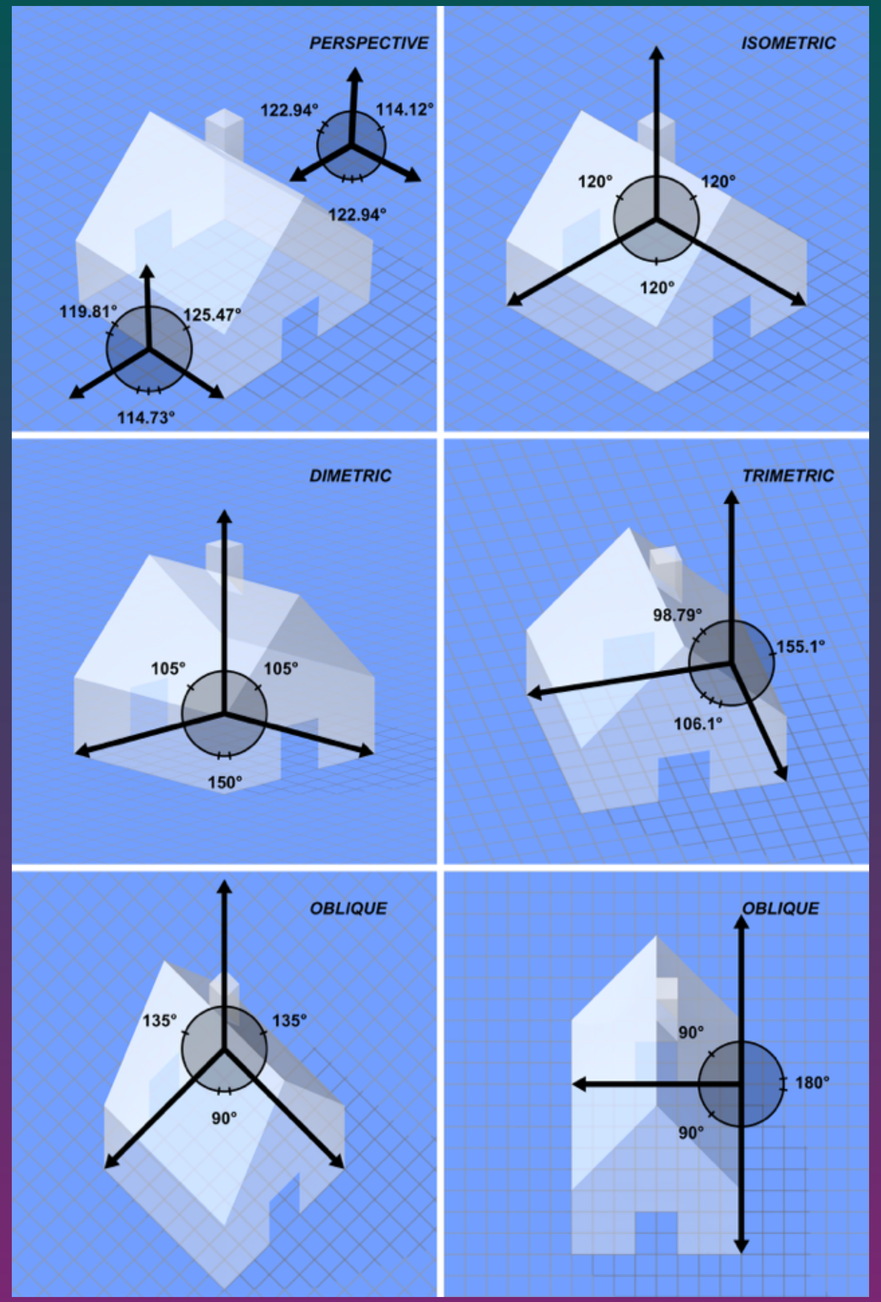


The view matrix is the INVERSE of our camera’s transformation matrix

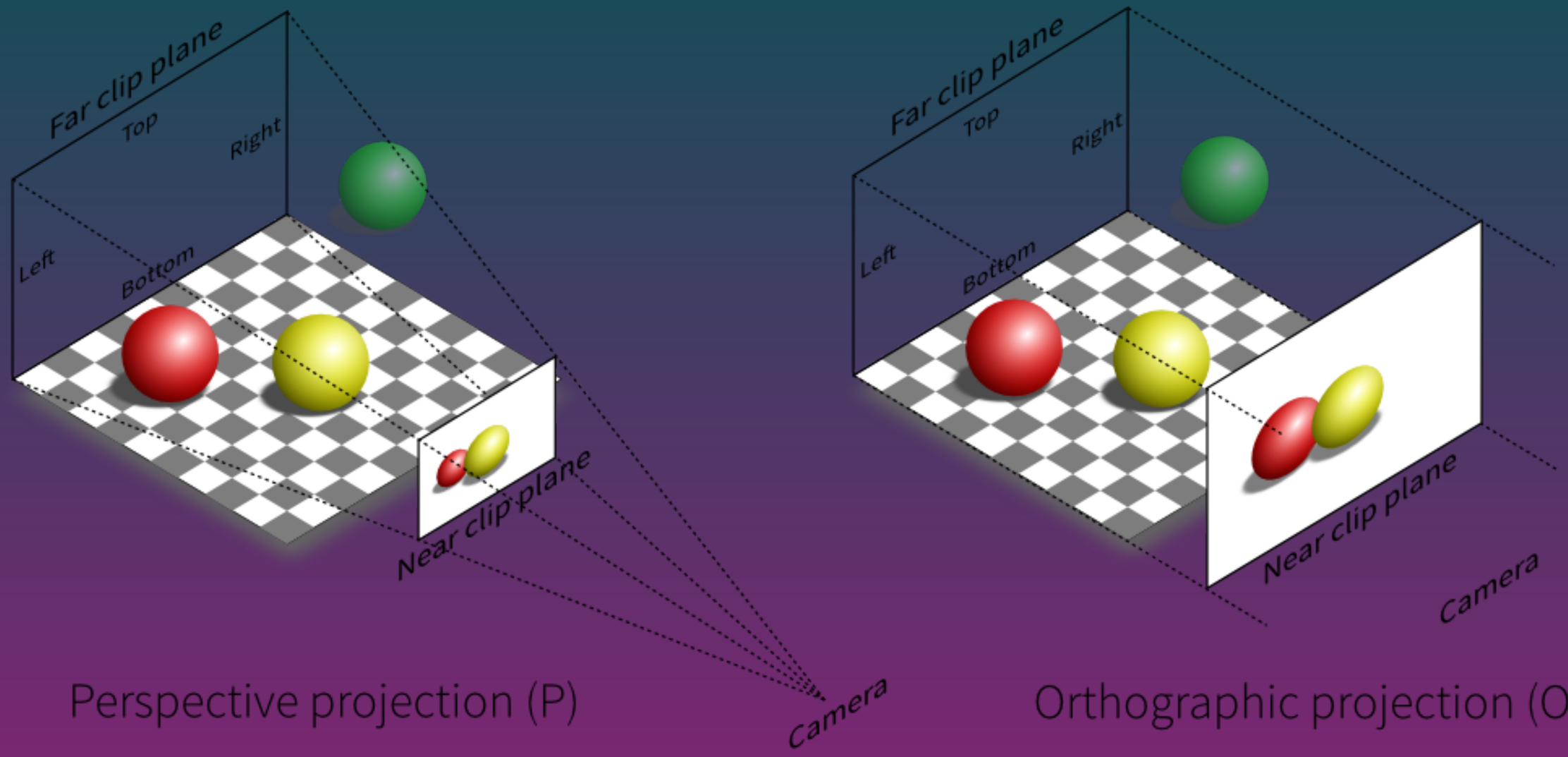
Projection matrix.

What is projection?

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



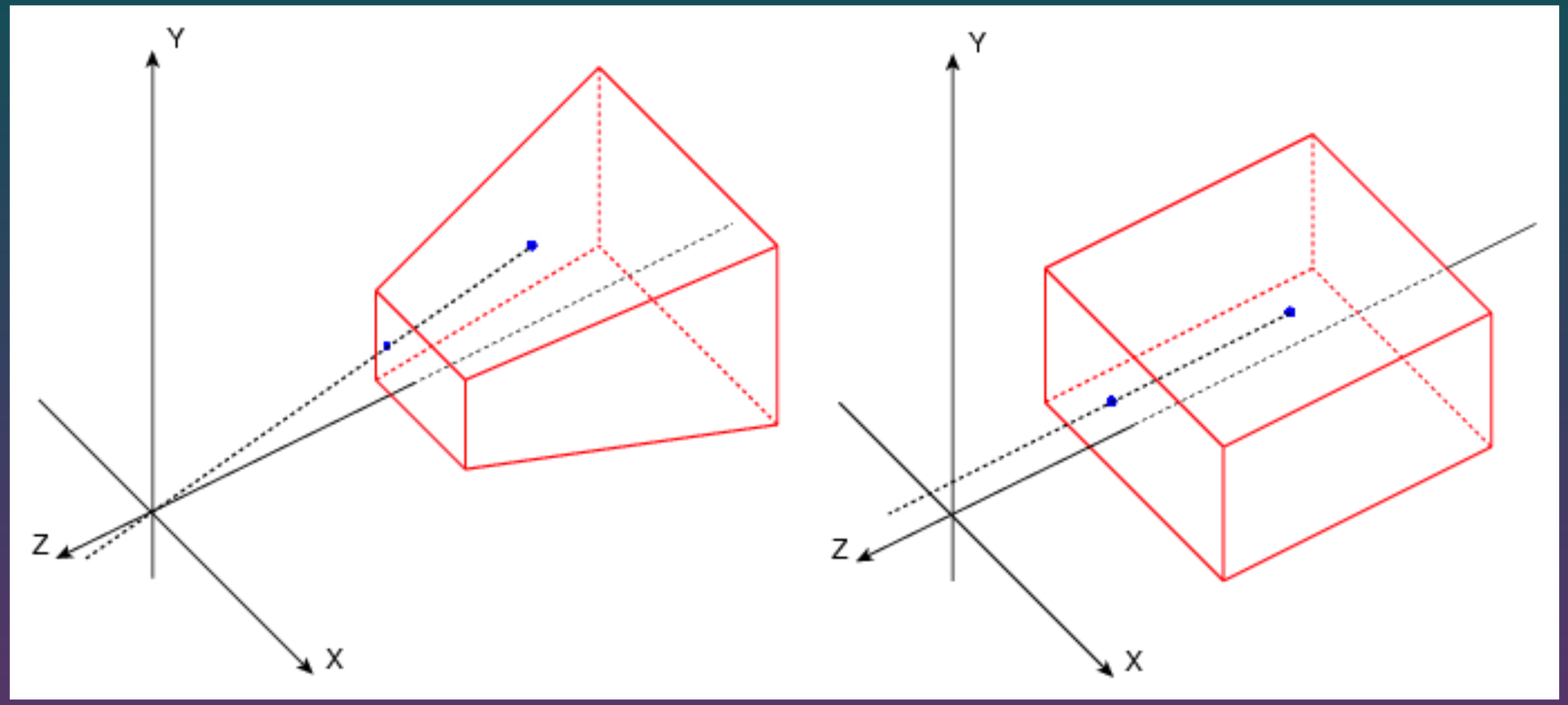
Perspective vs. Orthographic projection.

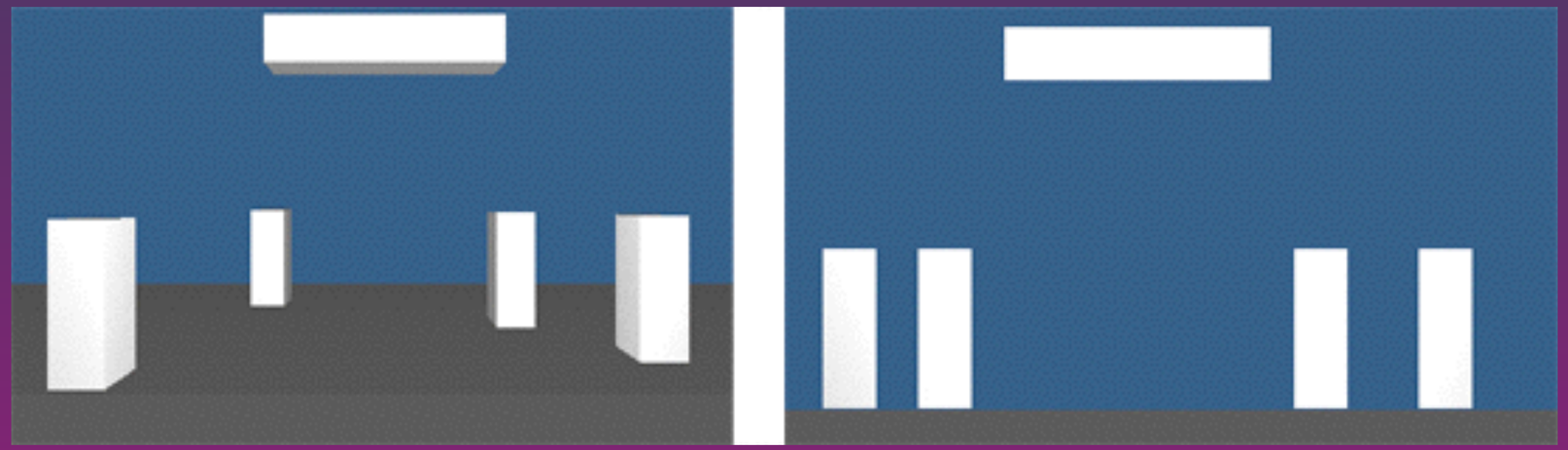


Perspective projection (P)

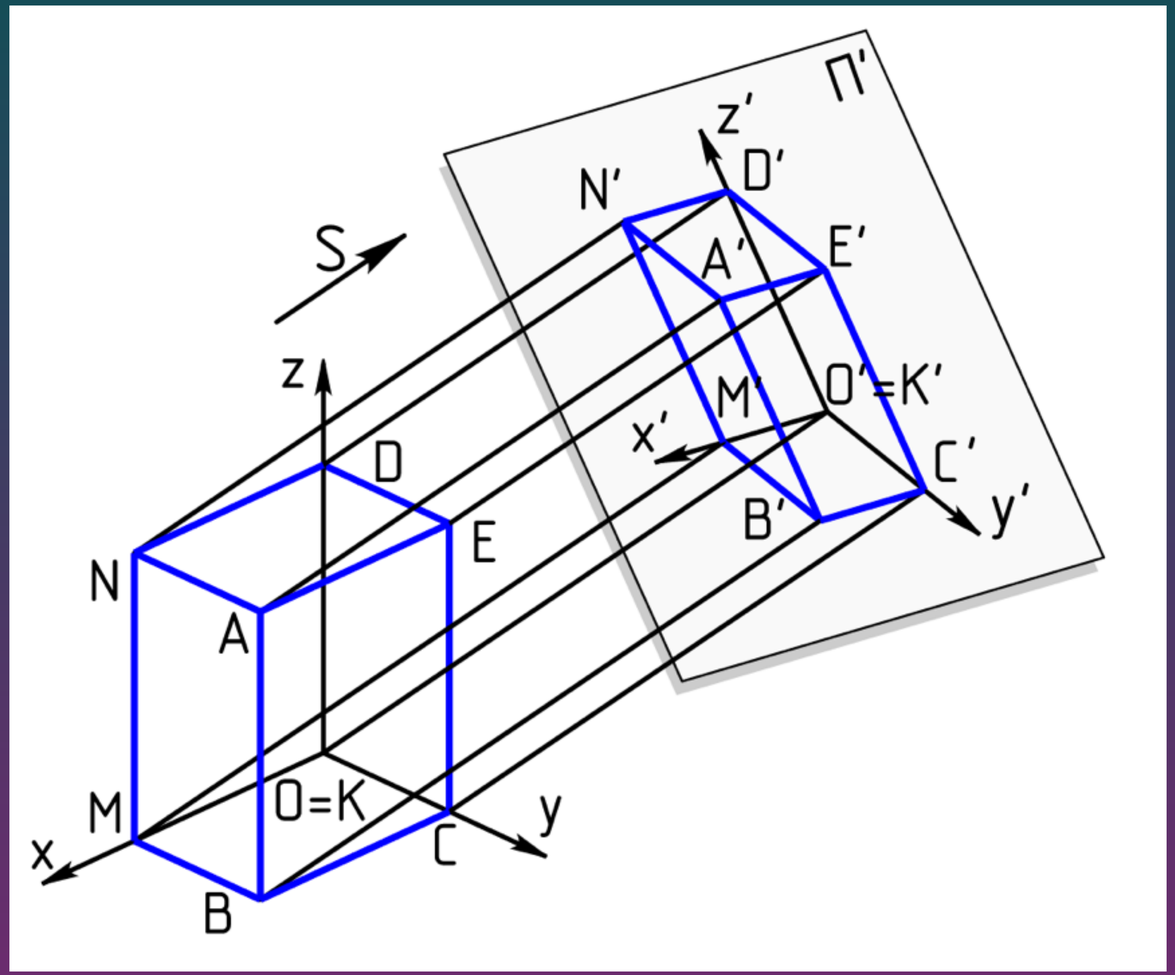
Orthographic projection (O)

Perspective vs. Orthographic projection.





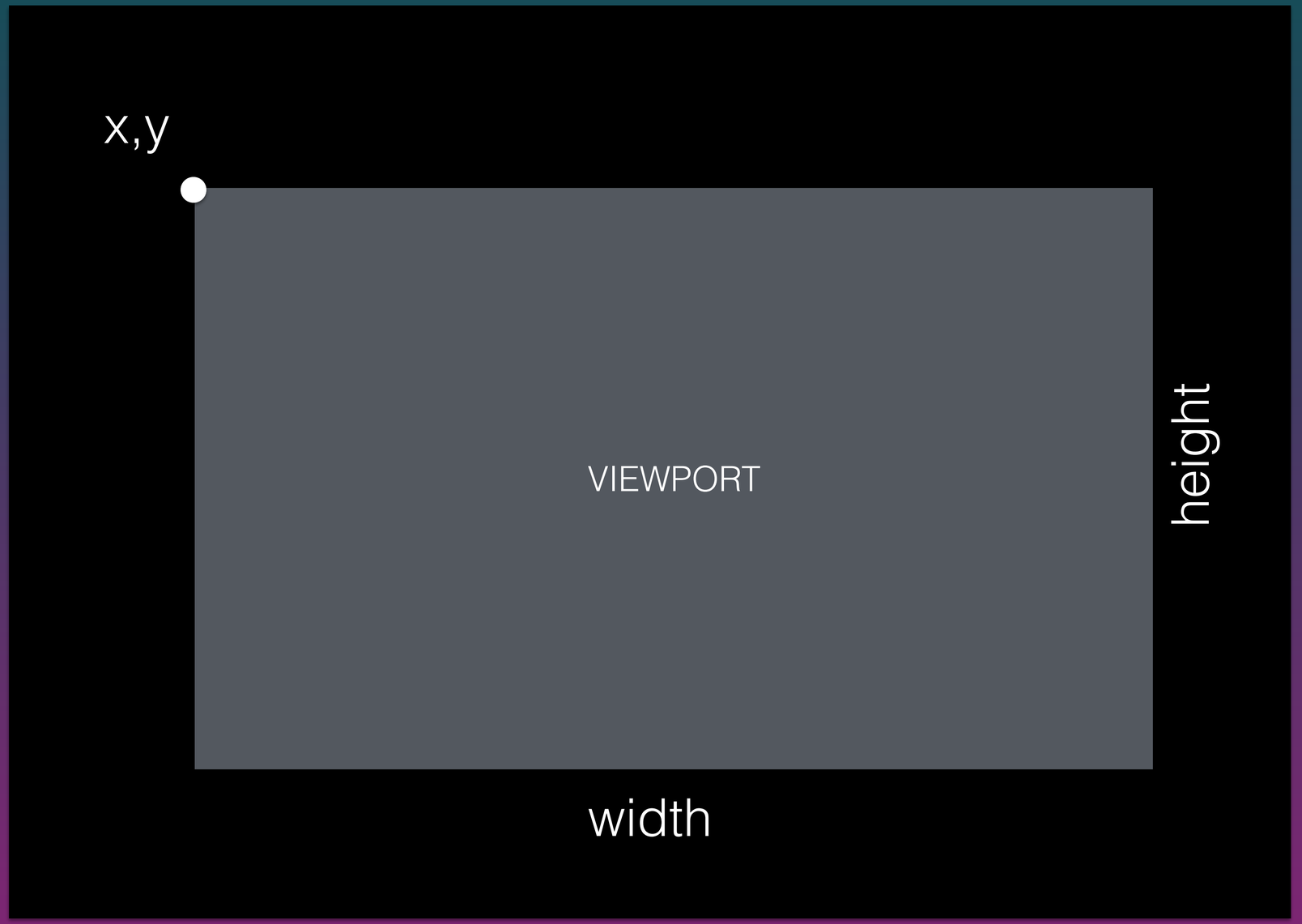
What is a projection matrix?



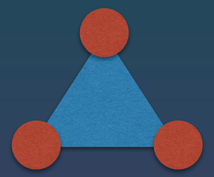
Vertex position * projection matrix = screen vertex position

Viewport mapping.

Offset and size of rendering area in pixels.



The rendering pipeline.



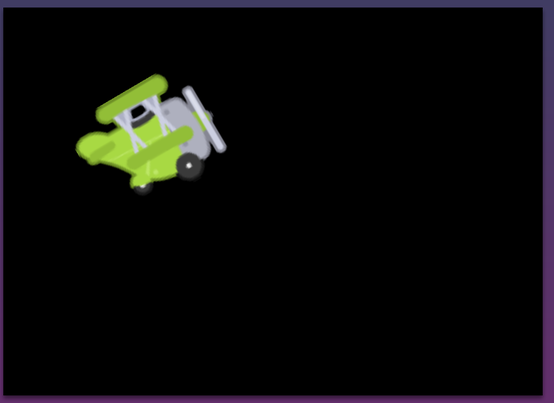
MODELVIEW MATRIX

PROJECTION MATRIX

VERTEX SHADER

VIEWPORT MAPPING

FRAGMENT SHADER



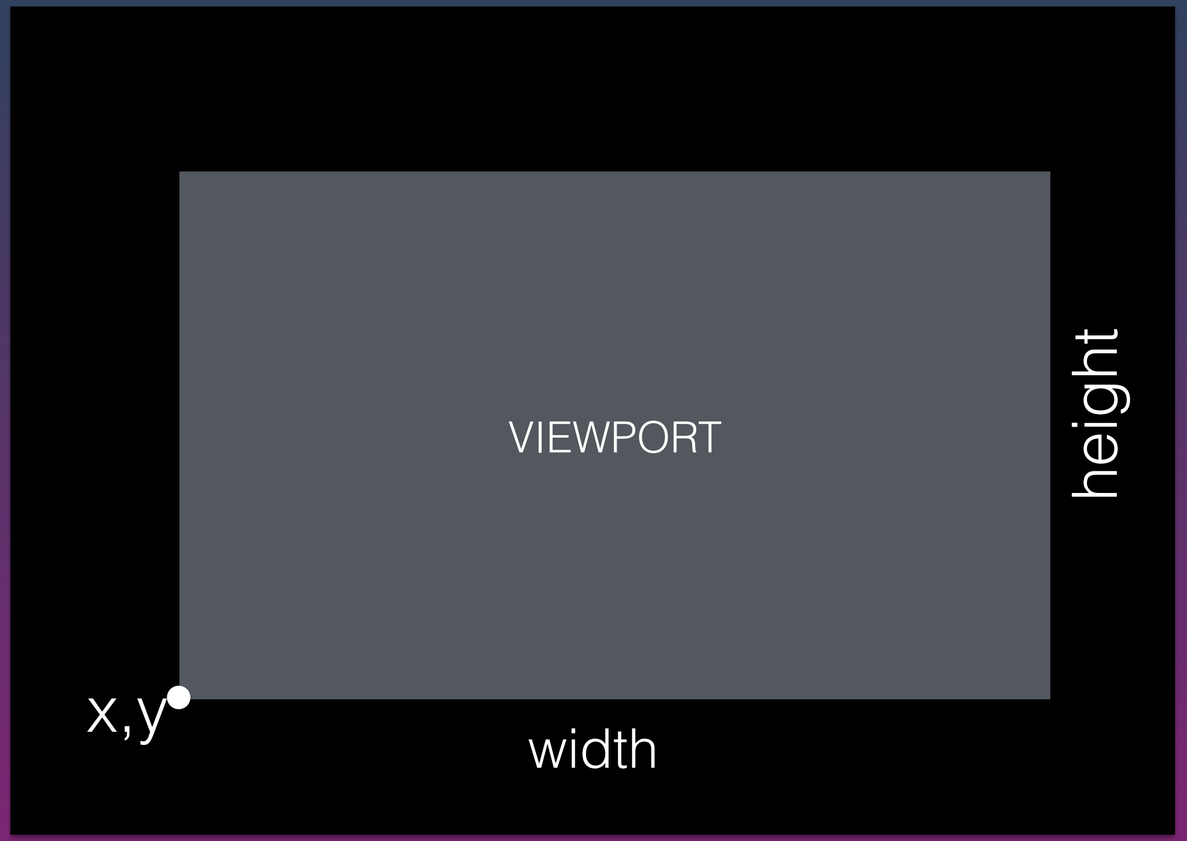

OpenGL®

The setup.

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

Sets the pixel size and offset of rendering area.

```
glViewport(0, 0, 800, 600);
```



```
glMatrixMode(matrixMode);
```

Selects the matrix that subsequent matrix operations will operate on.

```
glMatrixMode(GL_PROJECTION);
```

Selects the projection matrix.

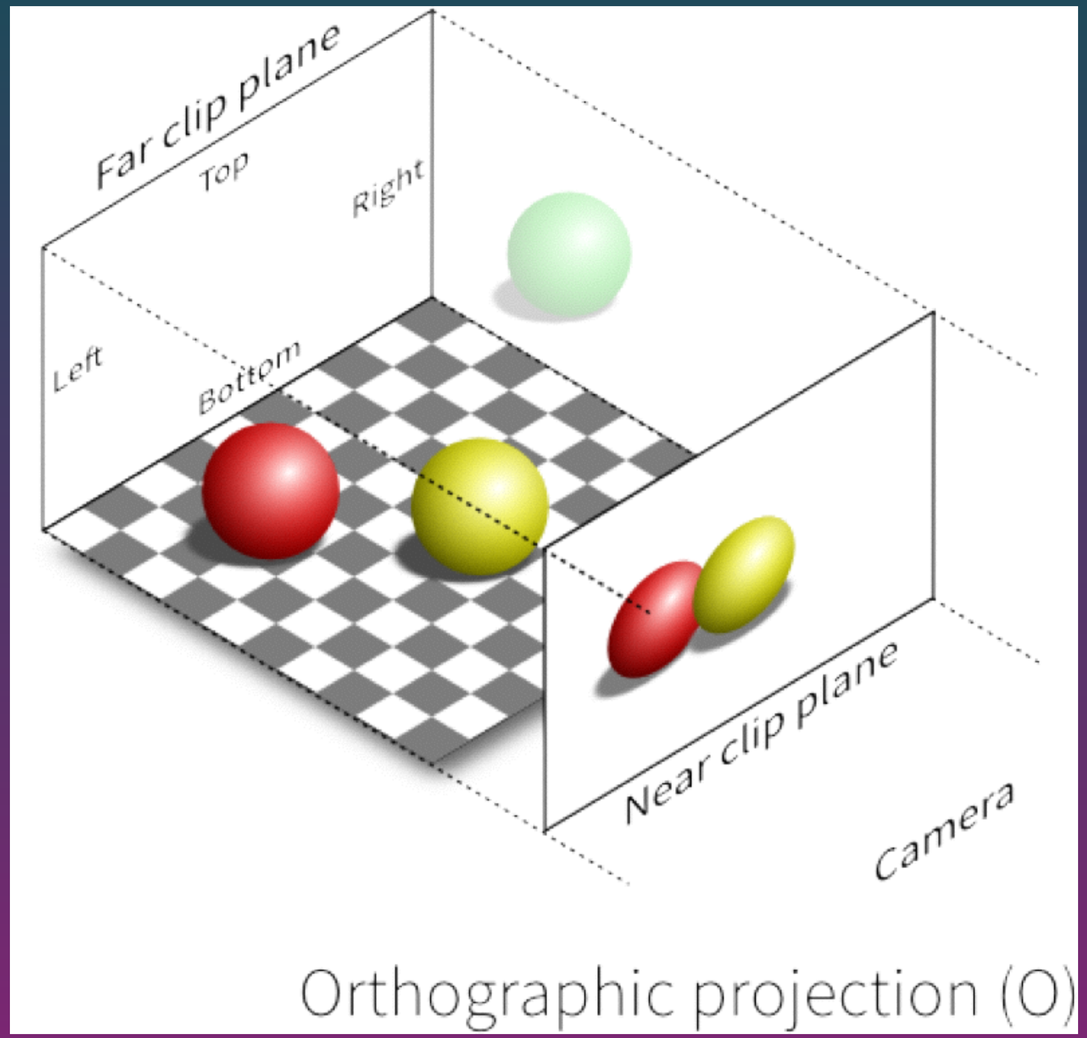
```
glMatrixMode(GL_MODELVIEW);
```

Selects the modelview matrix.

```
void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);
```

```
glOrtho(-1.33, 1.33, -1.0, 1.0, -1.0, 1.0);
```

Multiplies the currently selected matrix with an orthographic projection matrix.



Orthographic projection (O)

Drawing shapes.

```
glVertexPointer (GLint size, GLenum type,
GLsizei stride, const GLvoid *pointer);
```

Defines an array of vertex data.

```
GLfloat triangle[] = {0.0f, 0.5f, -0.5f, -0.5f, 0.5f, -0.5f};

glVertexPointer(2, GL_FLOAT, 0, triangle);
```

```
void glEnableClientState (GLenum array);
```

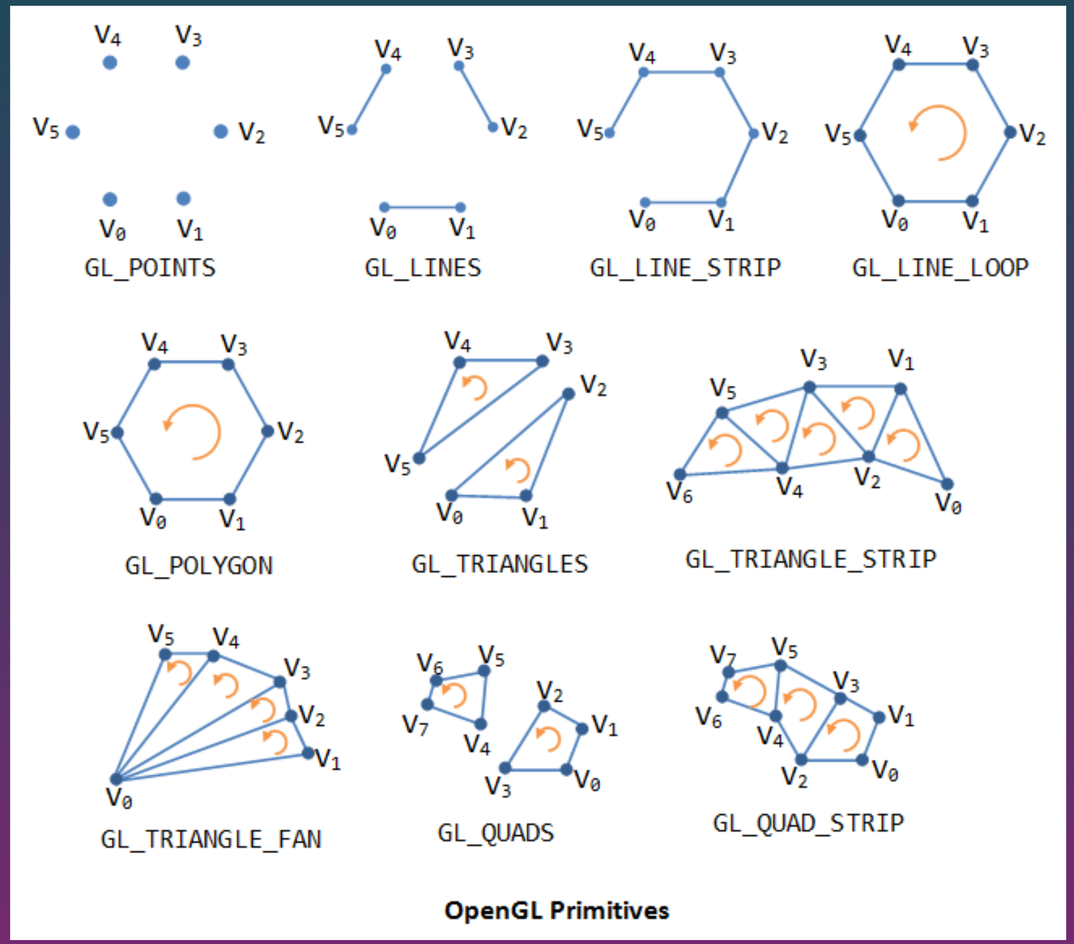
Enables a client state capability.

```
glEnableClientState(GL_VERTEX_ARRAY);
```

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

Render previously defined arrays.

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



Transformations.

```
void glMatrixMode(GLenum matrixMode);
```

Selects the matrix that subsequent matrix operations will operate on.

```
glMatrixMode(GL_PROJECTION);
```

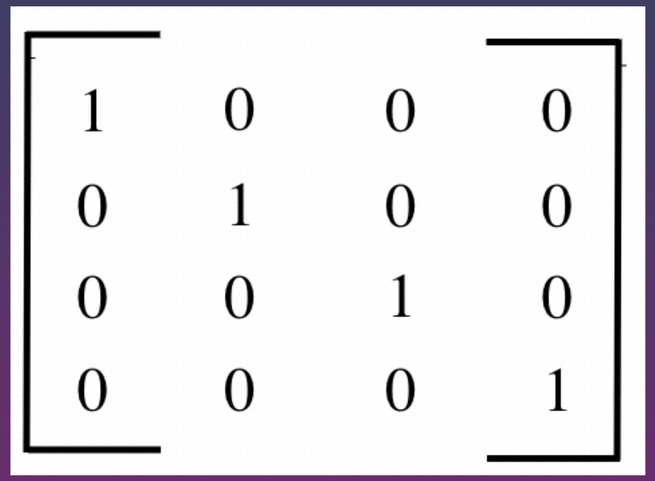
Selects the projection matrix.

```
glMatrixMode(GL_MODELVIEW);
```

Selects the model view matrix.

```
void glLoadIdentity();
```

Replaces the current matrix with the identity matrix.



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

The identity matrix results in no transformation to the vector it is multiplied by. It is the “reset state” for transformations.

```
void glTranslatef (GLfloat x, GLfloat y, GLfloat z);
```

Move. (Multiply the current matrix by a translation matrix).

```
void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
```

Rotate. (Multiply the current matrix by a rotation matrix).

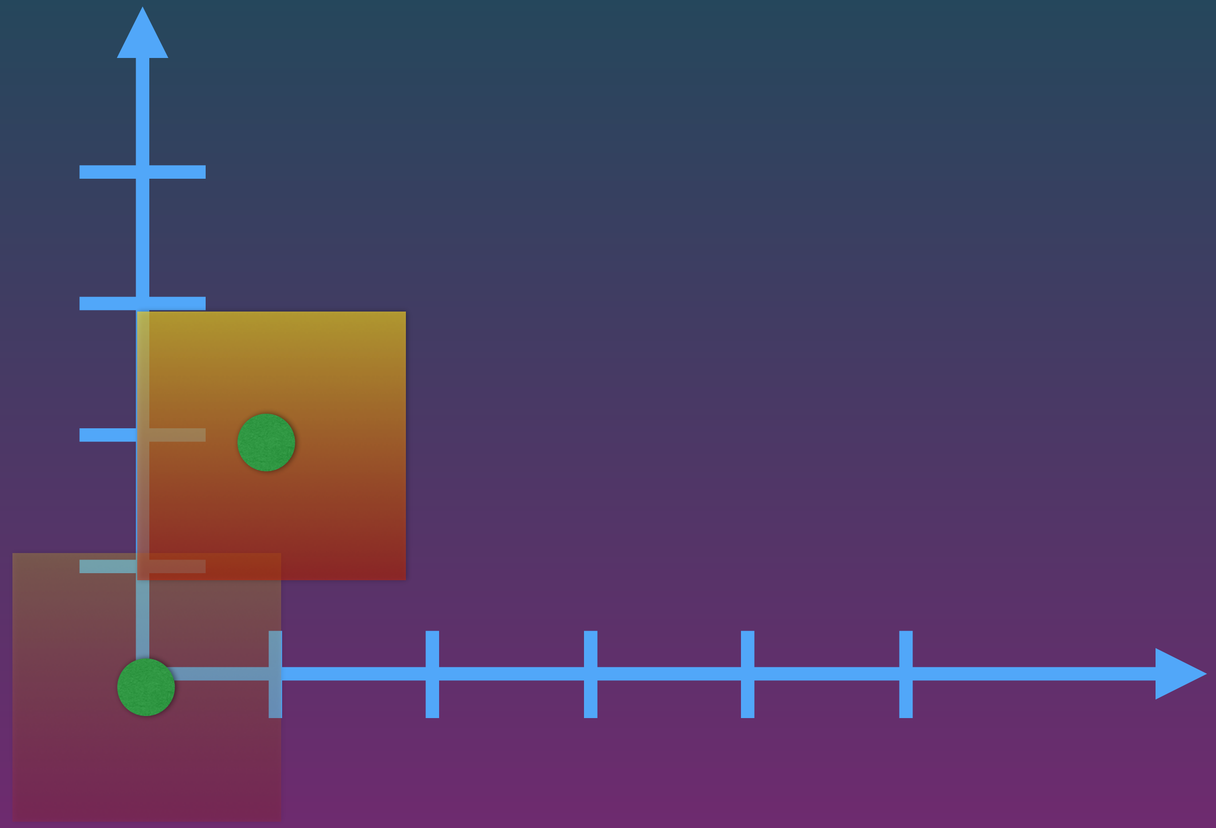
```
void glScalef (GLfloat x, GLfloat y, GLfloat z);
```

Scale. (Multiply the current matrix by a scale matrix).

```
void glTranslatef (GLfloat x, GLfloat y, GLfloat z);
```

Move. (Multiply the current matrix by a translation matrix).

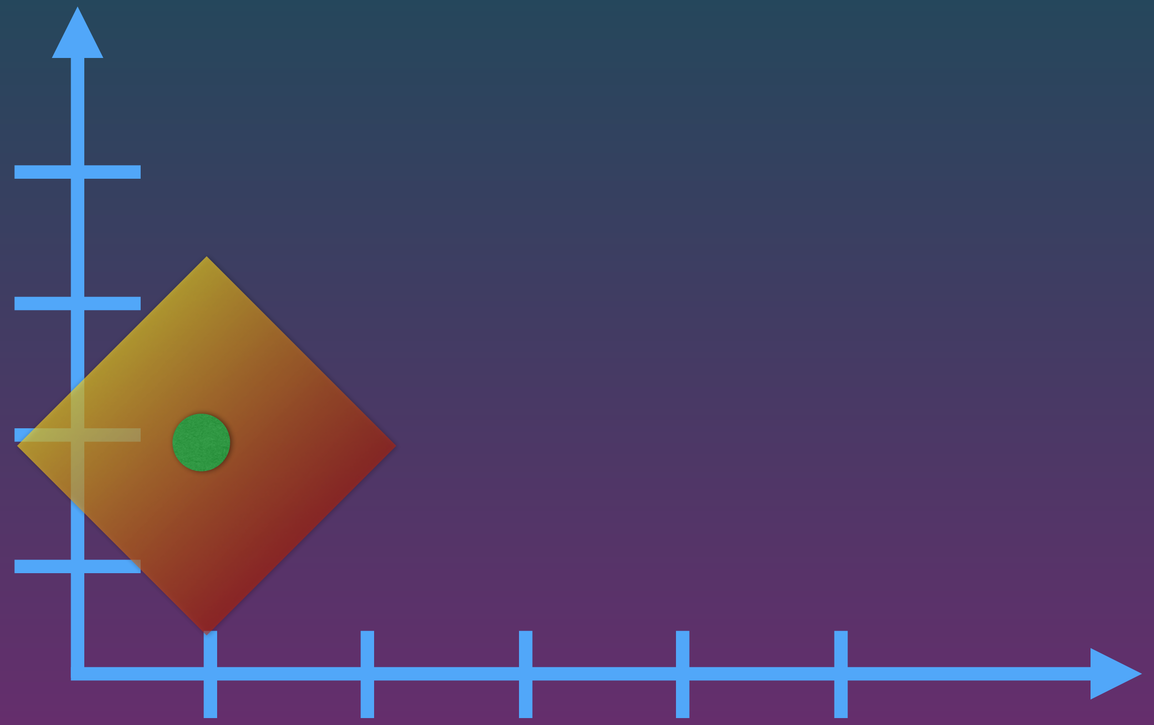
```
glLoadIdentity();
glTranslatef(1.0, 2.0, 0.0);
```



```
void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
```

Rotate. (Multiply the current matrix by a rotation matrix).

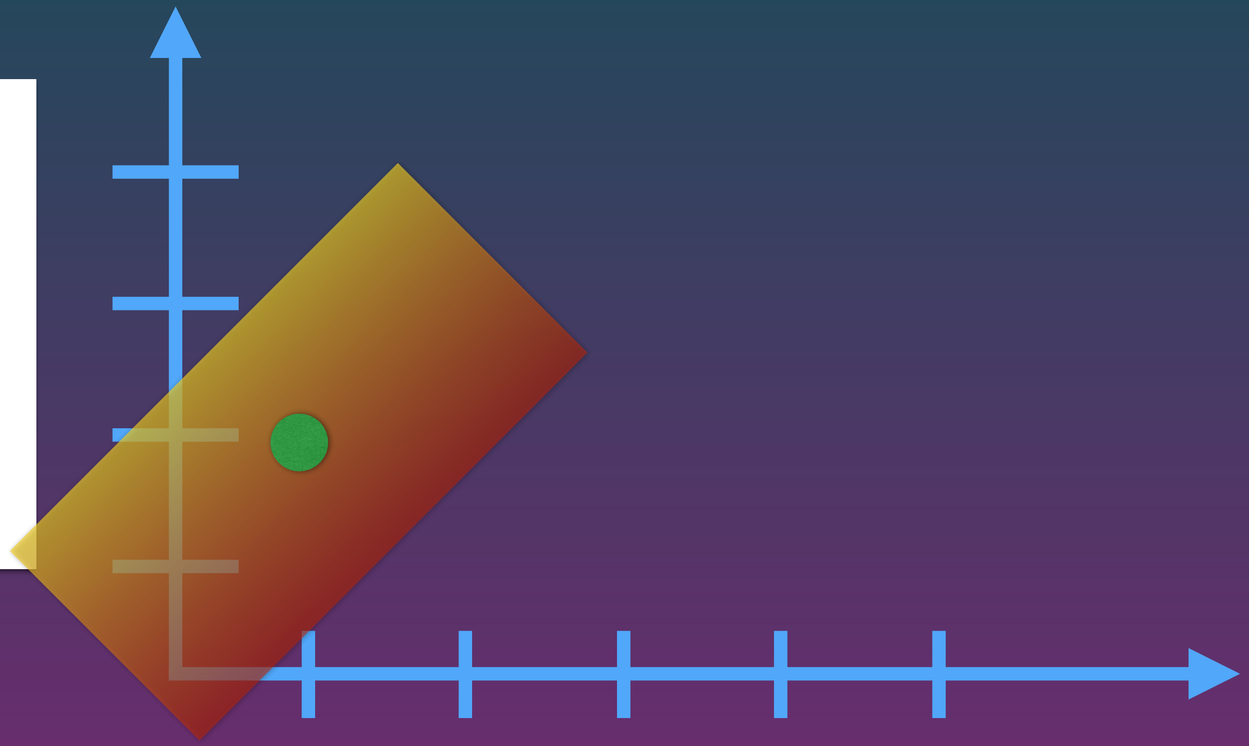
```
glLoadIdentity();
glTranslatef(1.0, 2.0, 0.0);
glRotatef(45.0f, 0.0, 0.0, 1.0);
```



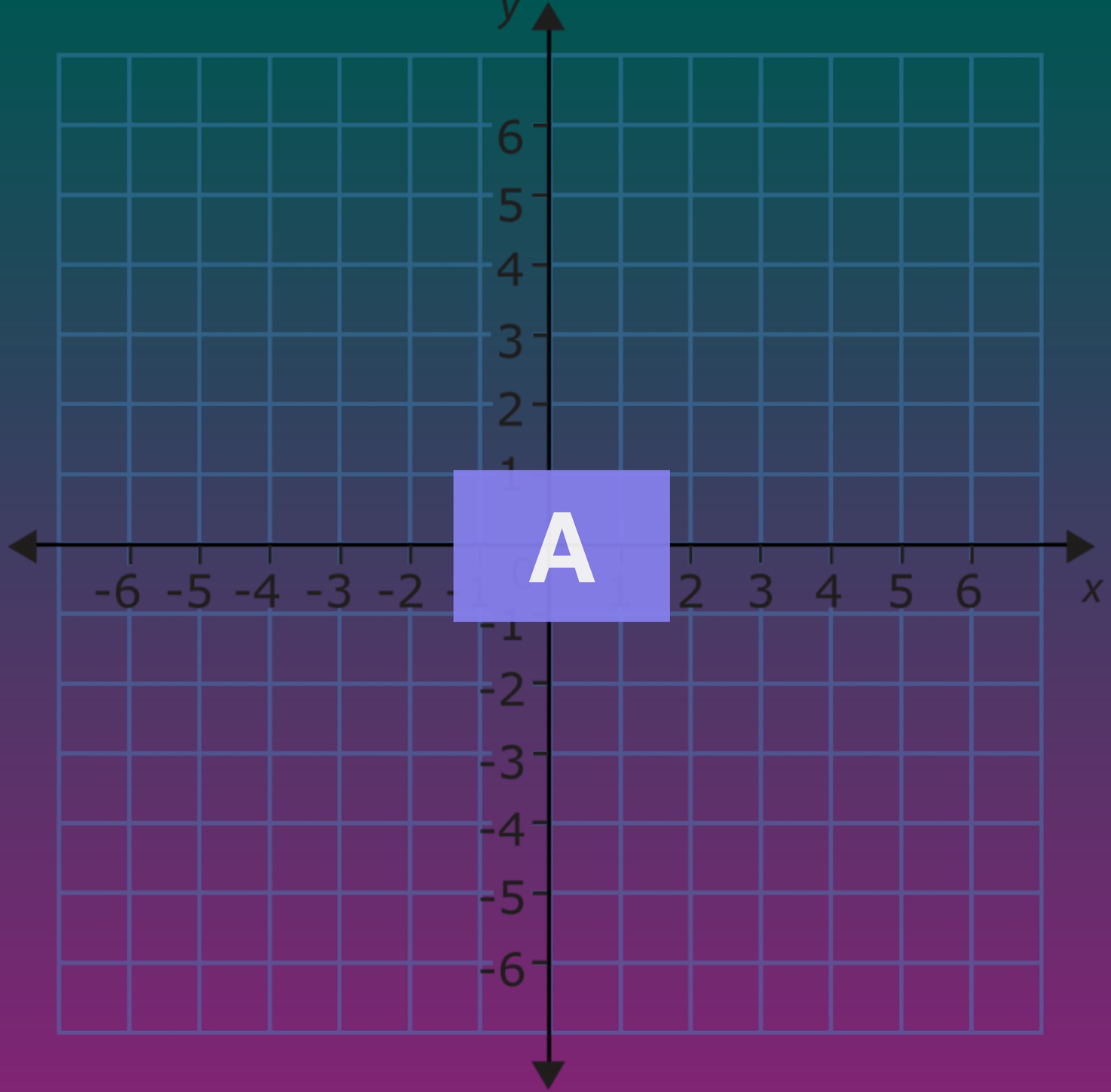
```
void glScalef (GLfloat x, GLfloat y, GLfloat z);
```

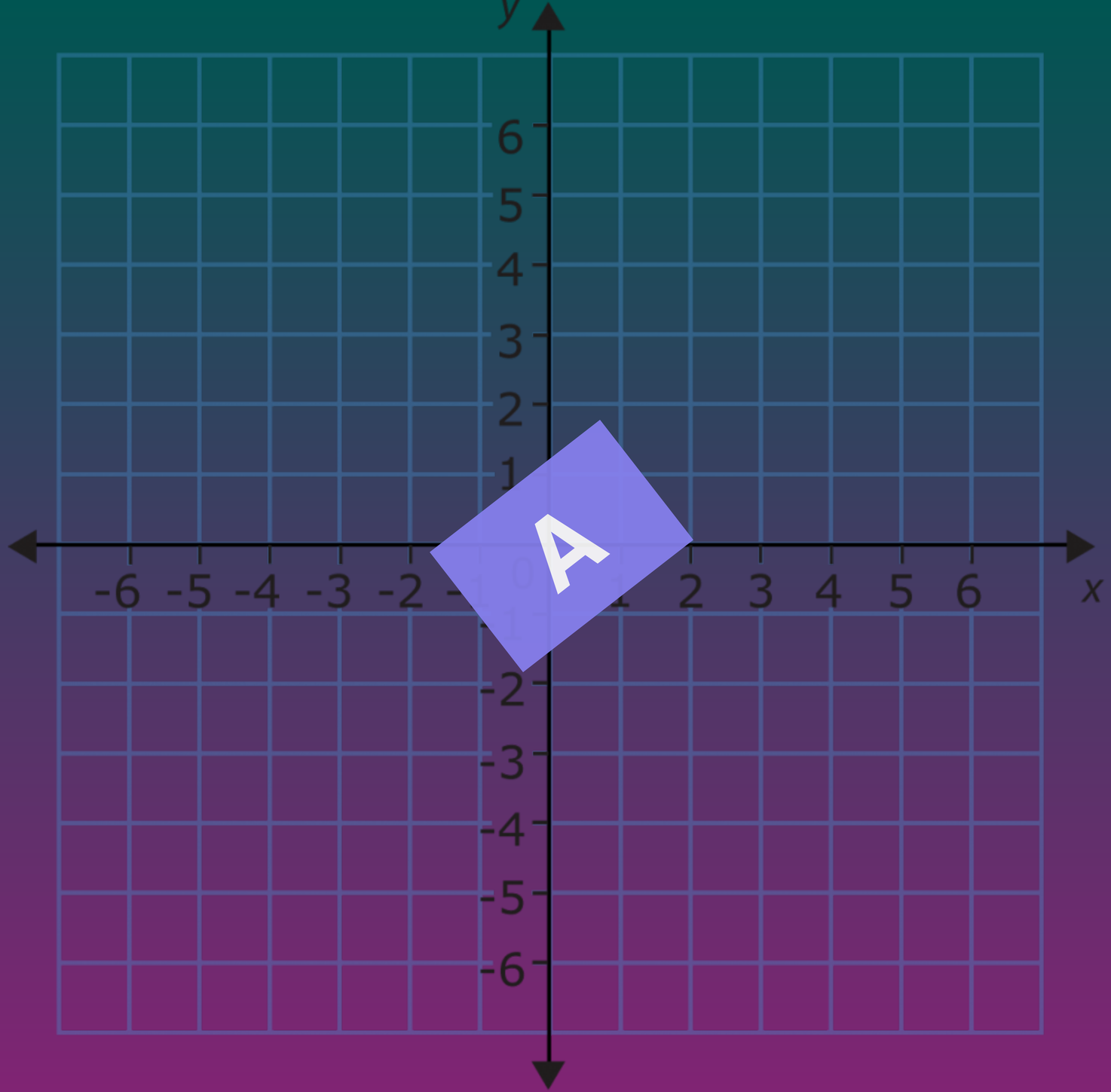
Scale. (Multiply the current matrix by a scale matrix).

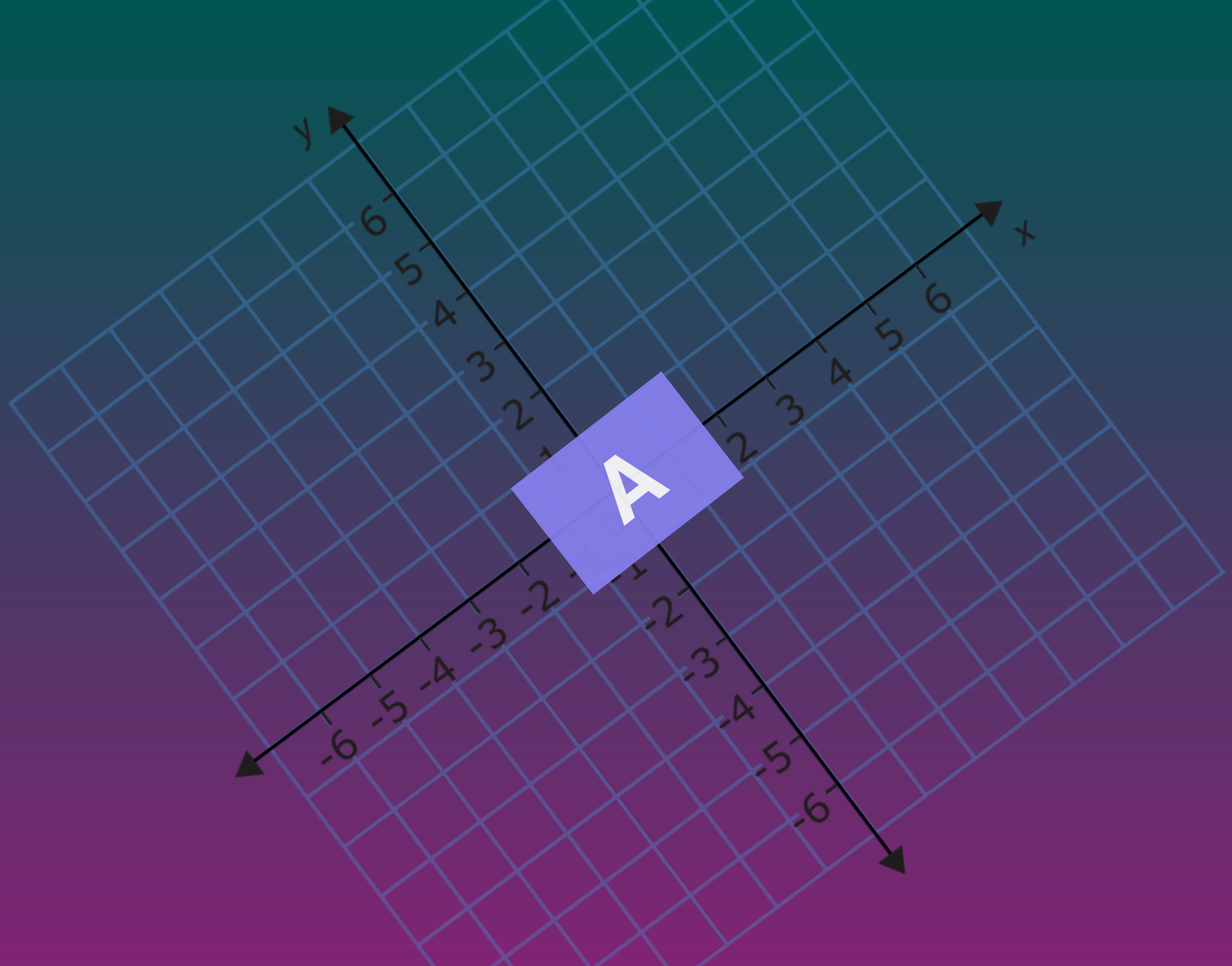
```
glLoadIdentity();
glTranslatef(1.0, 2.0, 0.0);
glRotatef(45.0f, 0.0, 0.0, 1.0);
glScalef(2.0, 1.0, 1.0);
```



Matrix operations are additive.


y
x
6
5
4
3
2
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2
2 3 4 5 6
A


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


y
x
A
6
5
4
3
2
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
2
3
4
5
6