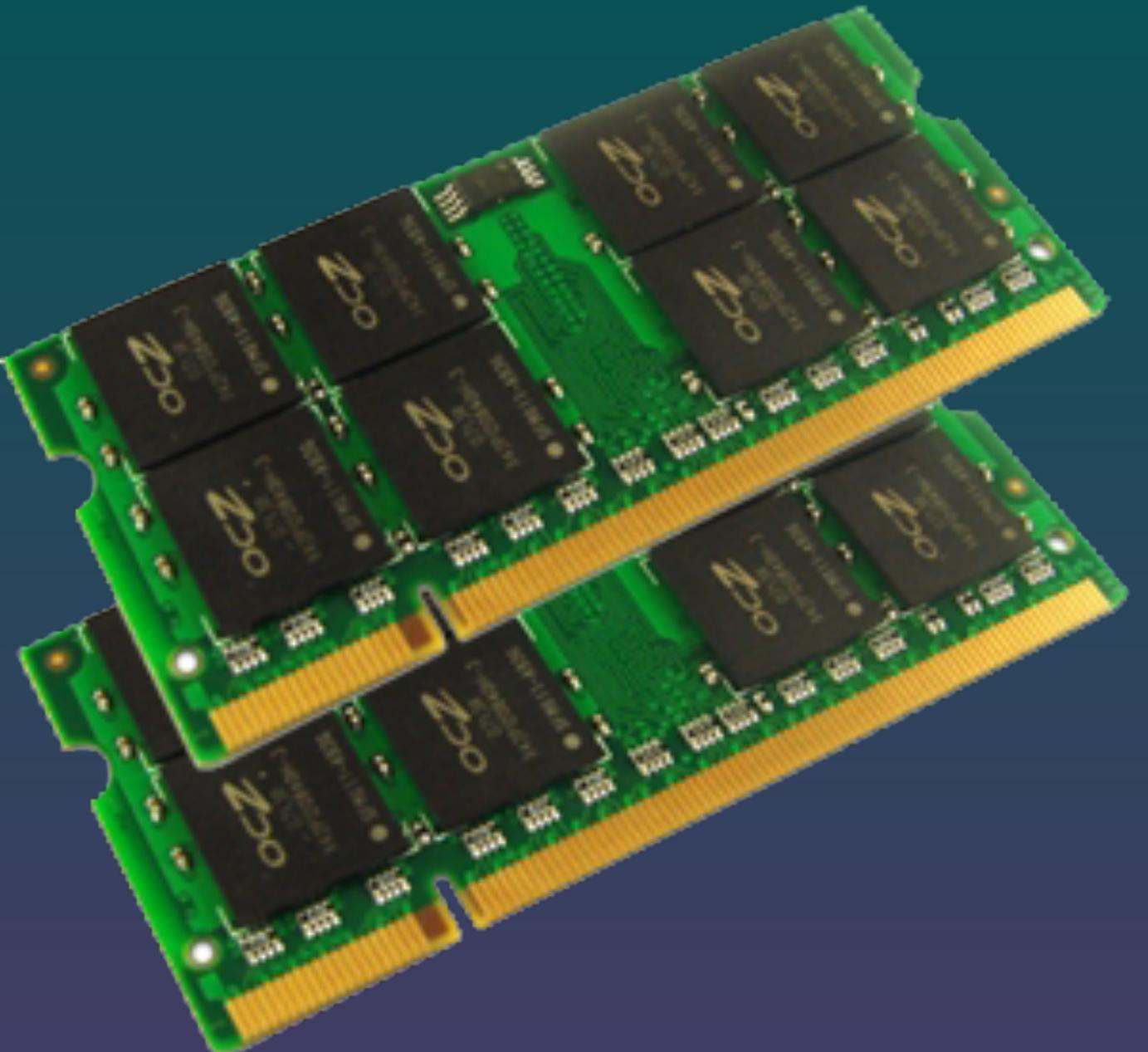


# Graphics Foundations



# Quick pointer refresher



000000008  
666666666  
866666666  
799997  
966666666  
799997  
...  


1	0	0	1
0	1	0	1

1 2 4 5 6 7 8 9 ...

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



1 2 4 5 6 7 8 9 ...

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



1 2 4 5 6 7 8 9 ...

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



1 2 4 5 6 7 8 9 ...

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



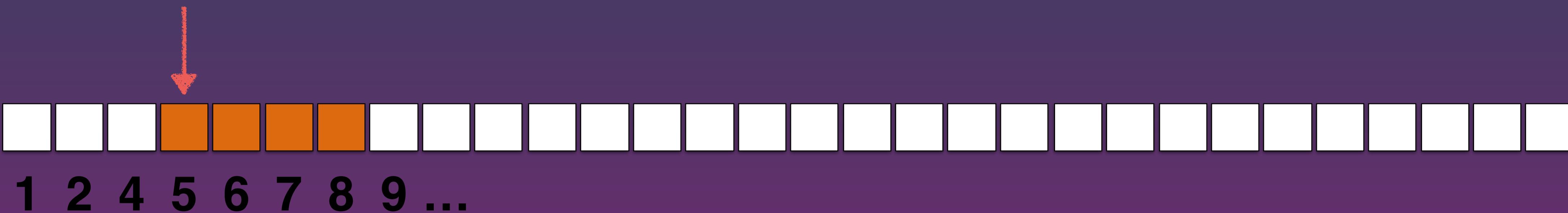
```
float *myVarPtr = &myVar;  
cout << myVarPtr << endl; // prints 5
```



```
float myVar = 16.0f;  
  
float *myVarPtr = &myVar;  
  
cout << myVarPtr << endl; // prints 5  
  
cout << *myVarPtr << endl; // prints 16  
  
cout << &myVar << endl; // prints 5
```

Dereference

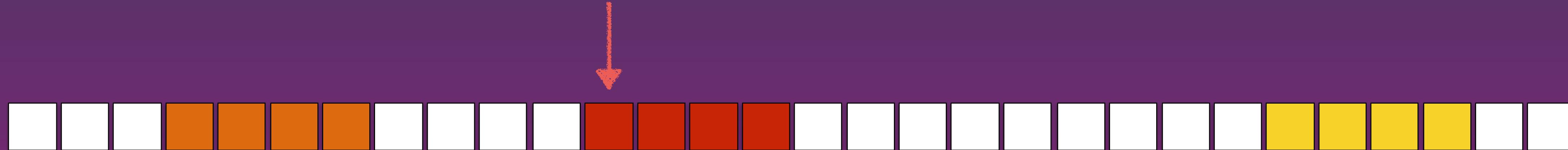
Reference



```
float myVar = 16.0f;  
  
float *myVarPtr = &myVar;  
  
cout << myVarPtr; // prints 5  
  
cout << *myVarPtr; // prints 16  
  
cout << &myVar; // prints 5  
  
float **myVarPtrPtr = &myVarPtr;  
  
cout << myVarPtrPtr; // prints 13
```

Dereference

Reference



1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

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

```
cout << myArray; // prints 5
```

```
cout << &myArray[0]; // prints 5
```

```
float *myArrayAsAPointer = myArray;
```

```
cout << myArrayAsAPointer; // prints 5
```

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

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

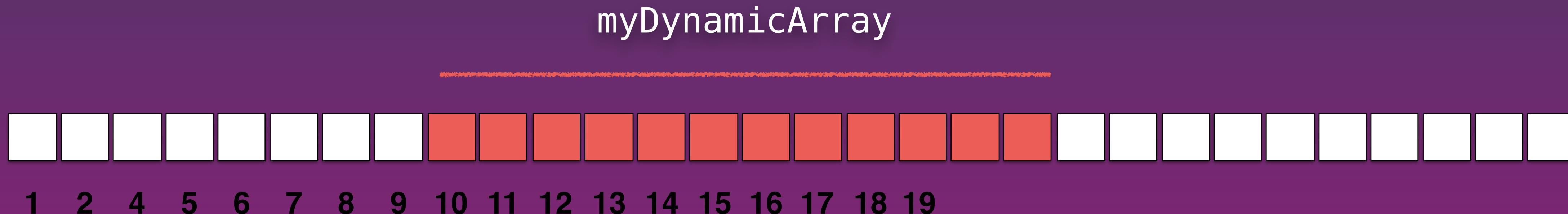
myArray

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



1 2 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

```
float *myDynamicArray;  
  
cout << myDynamicArray; // 0x0 or some uninitialized location  
cout << myDynamicArray[0]; // crash or garbage!  
  
myDynamicArray = new float[3];  
  
cout << myDynamicArray; // prints 10  
cout << myDynamicArray[0]; no crash!  
  
delete myDynamicArray;
```



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

myArray

---

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



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

myCharArray

---

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

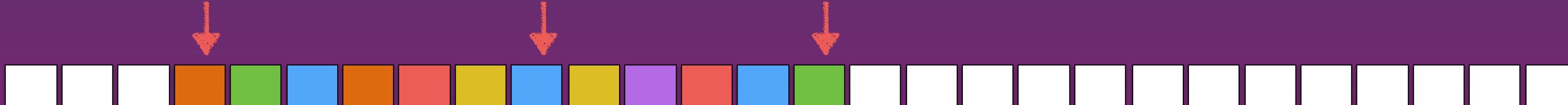


Image on the screen

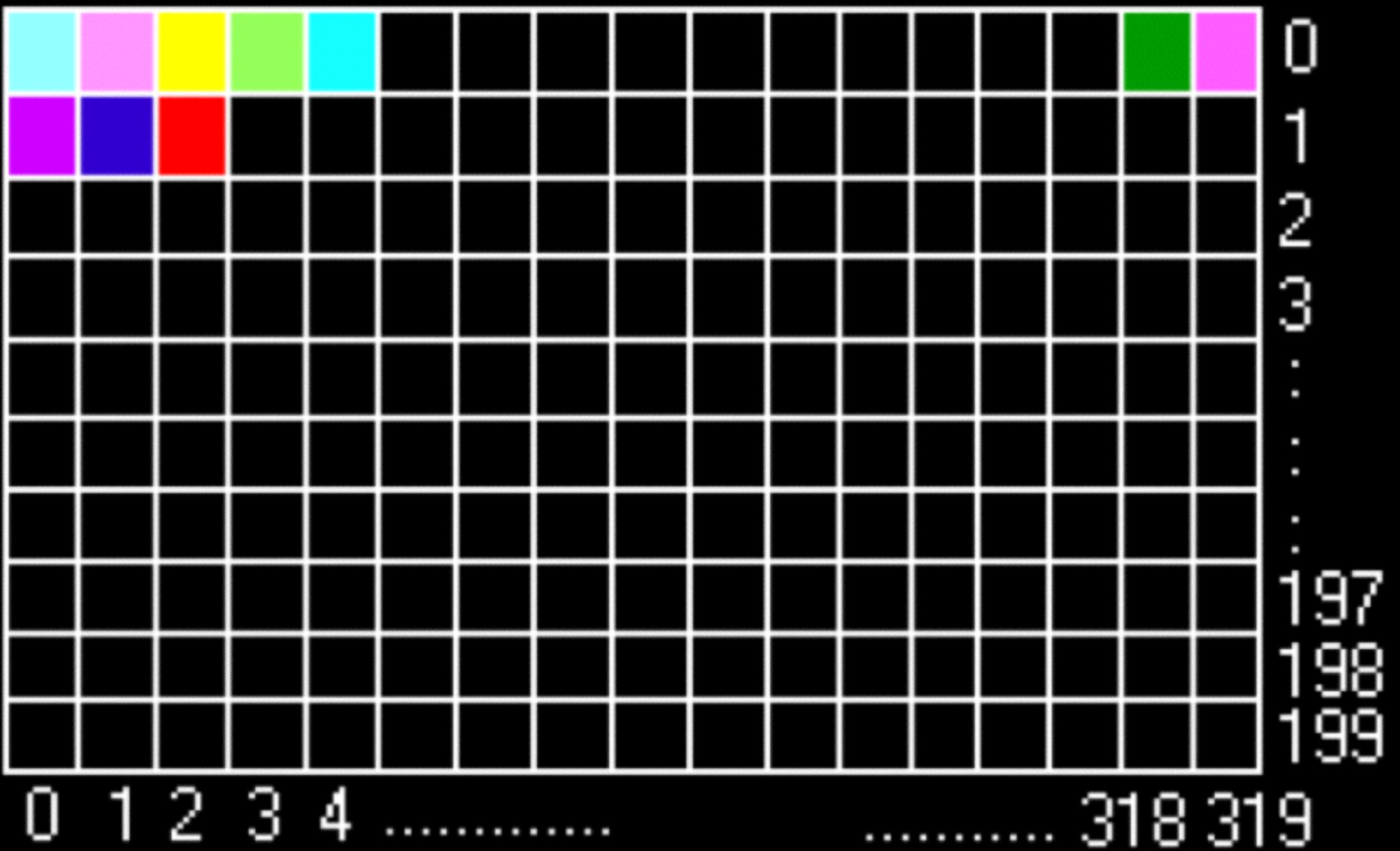
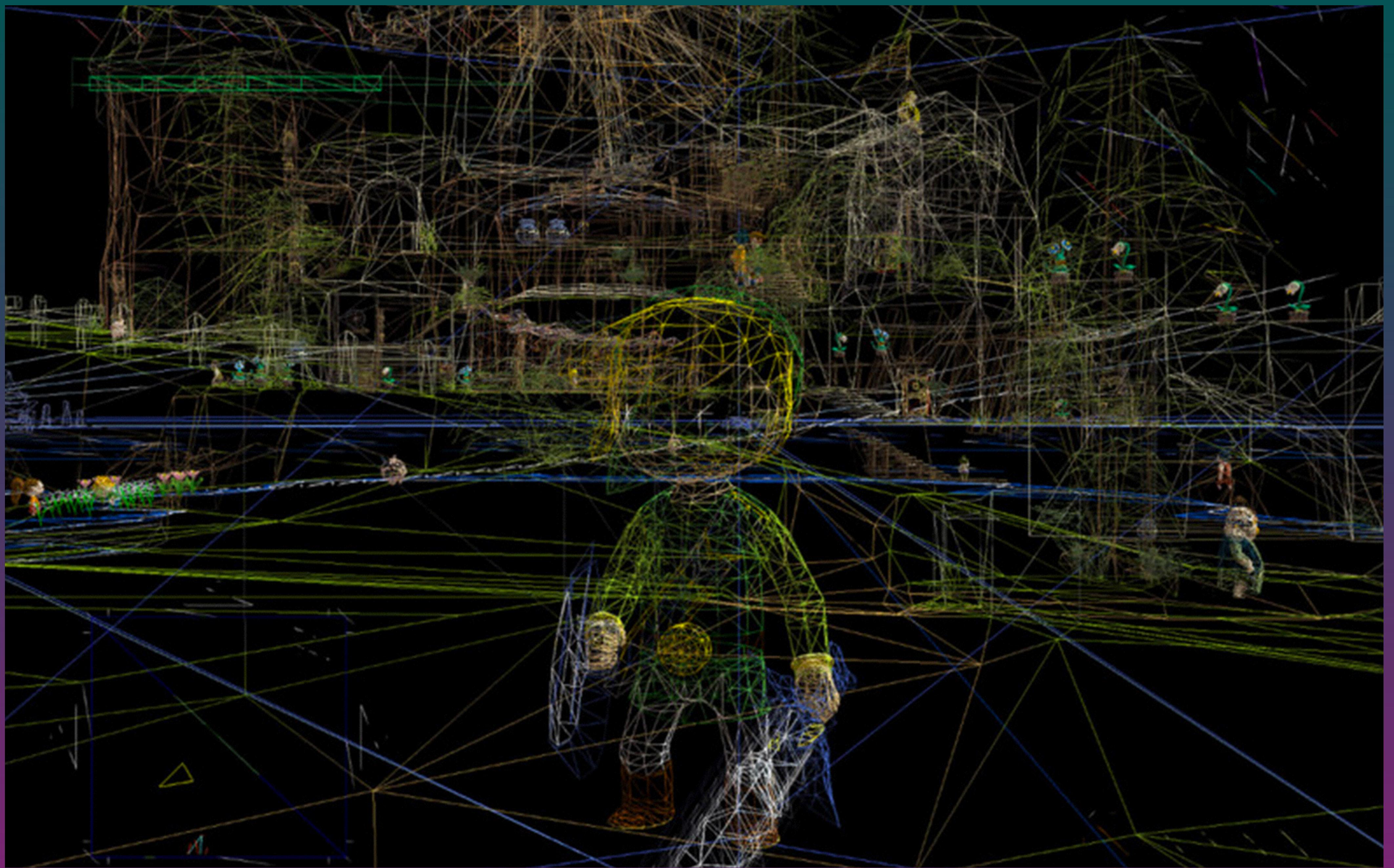


Image in memory



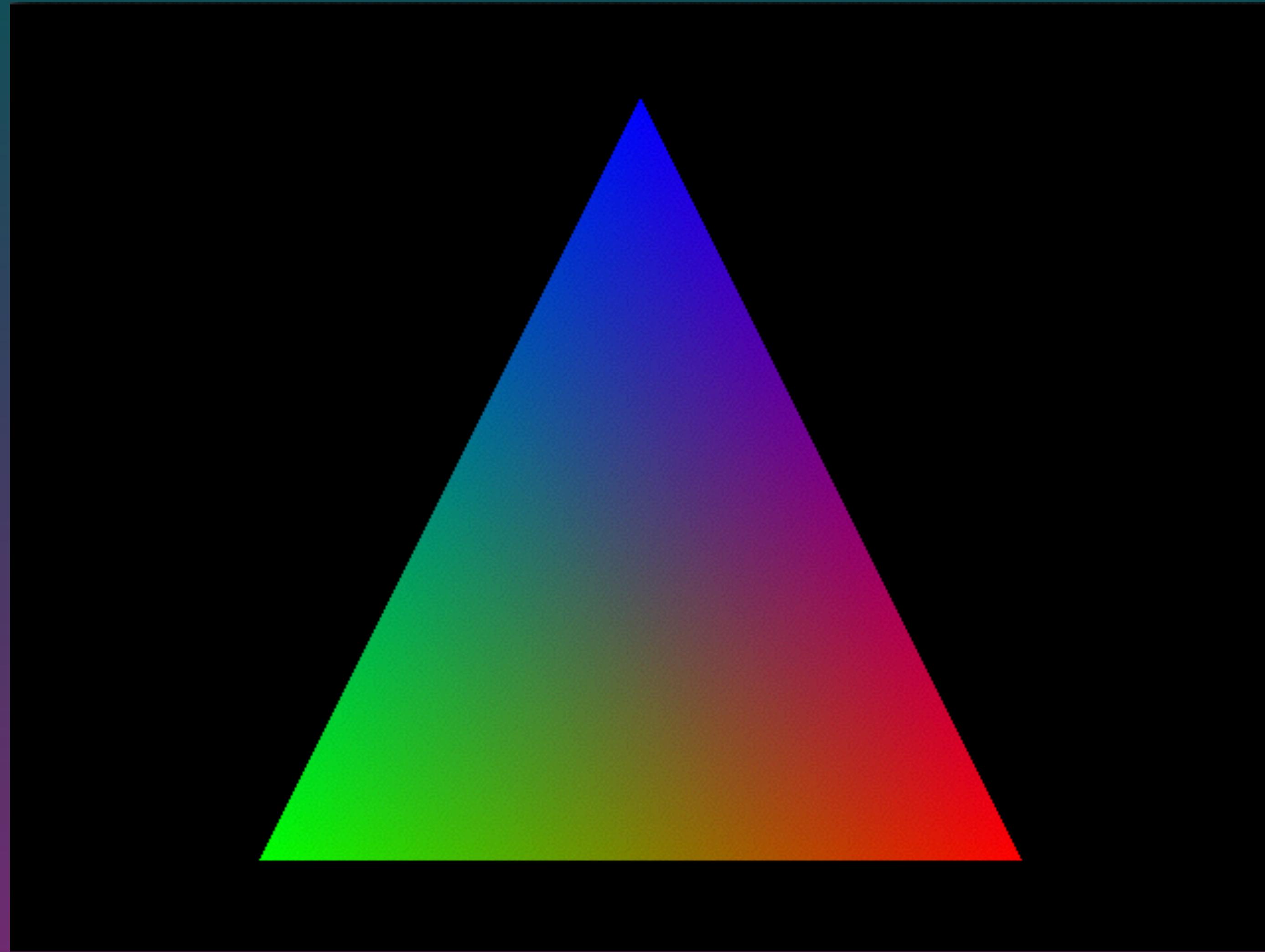
# Graphics in games

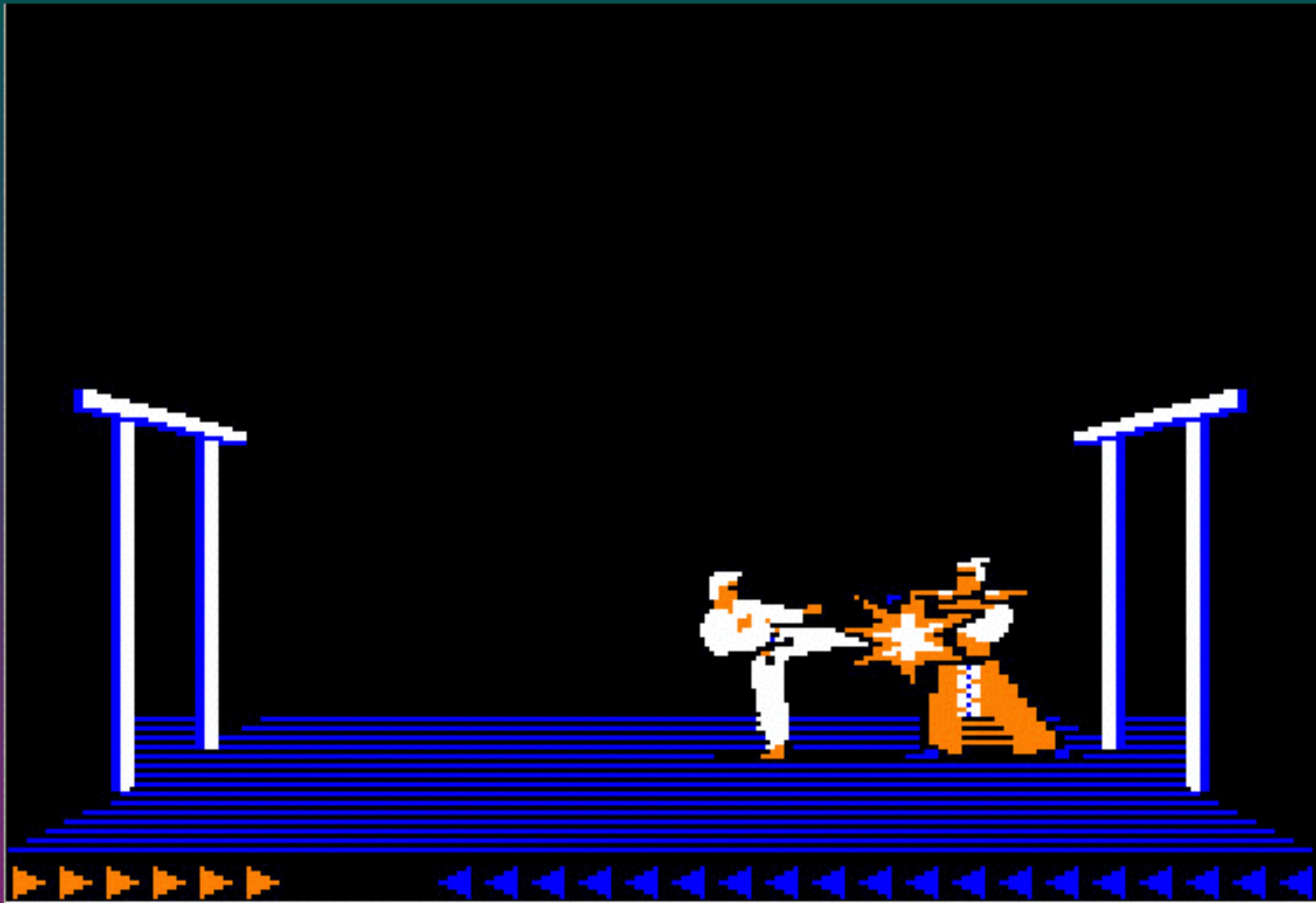




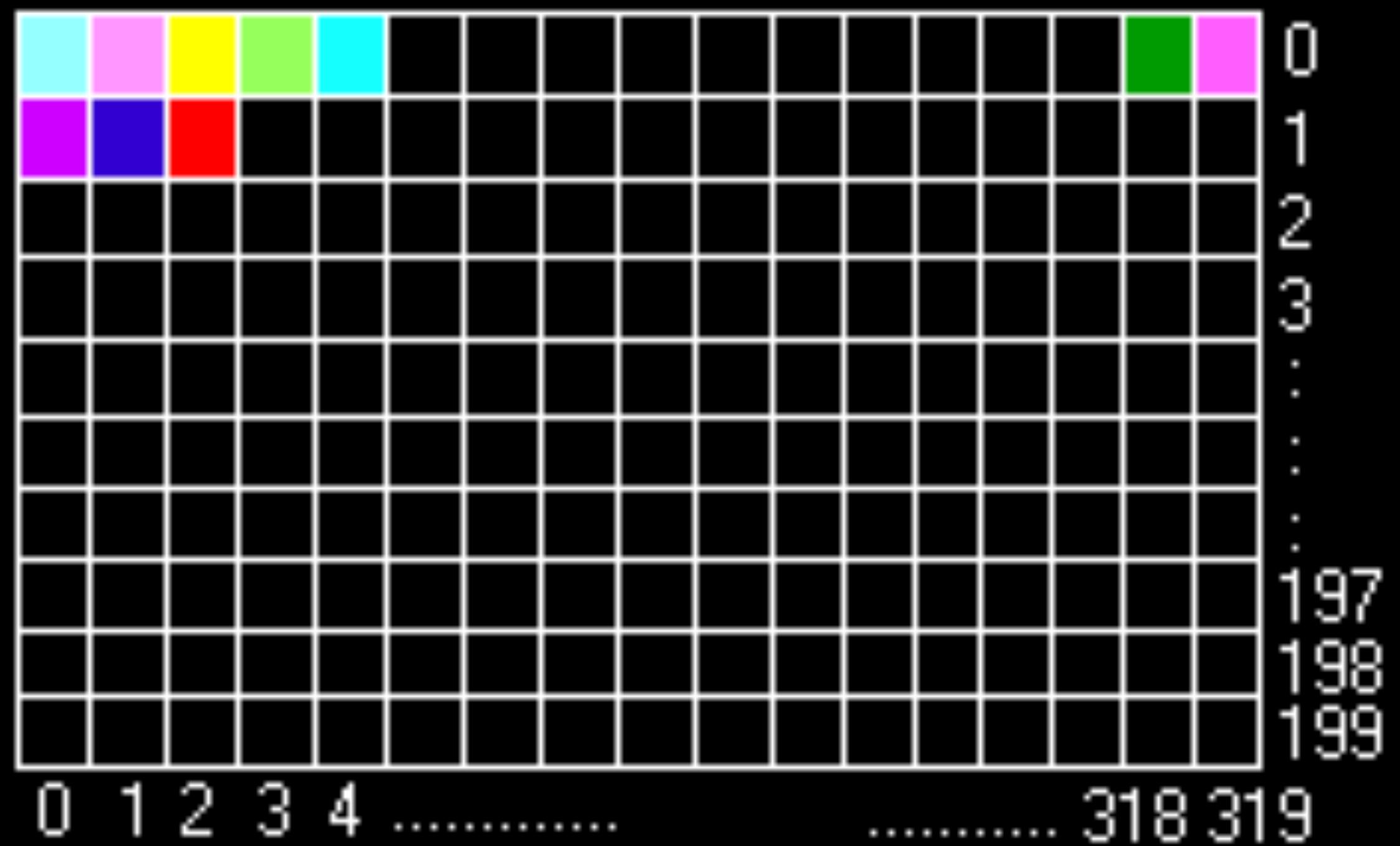








## The screen



## The video memory





MARIO  
000000

0x00

WORLD 1-1 TIME

# SUPER MARIO BROS.

©1985 NINTENDO

1 PLAYER GAME

2 PLAYER GAME

TOP - 000000

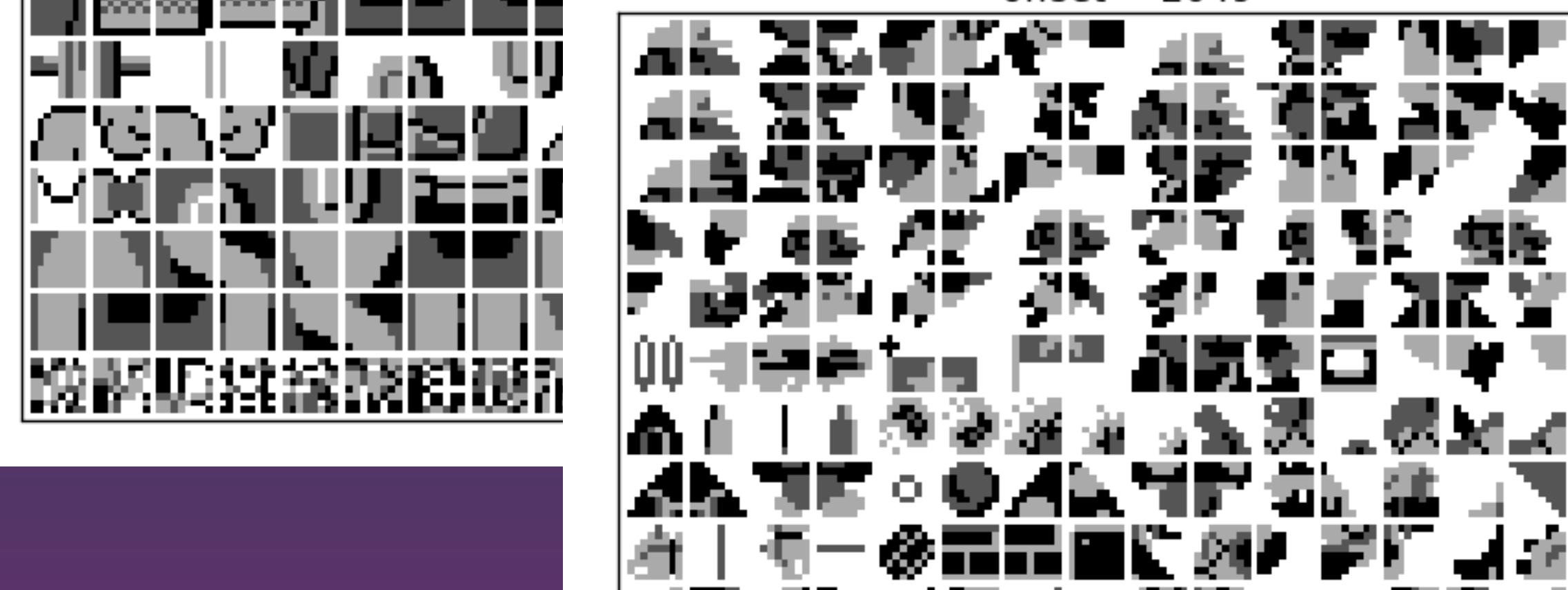


offset = 2305



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



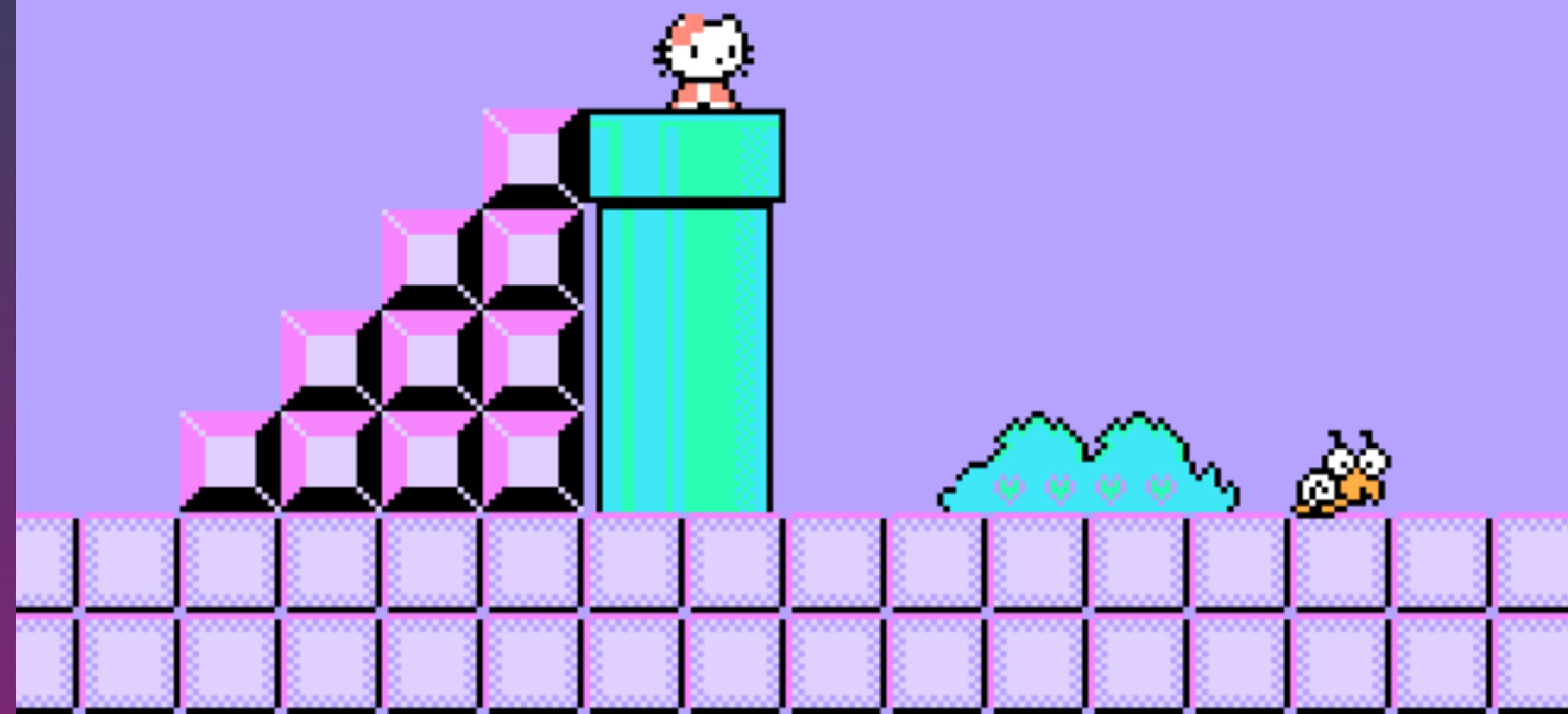
ロフア ( 10204050800 1UP )

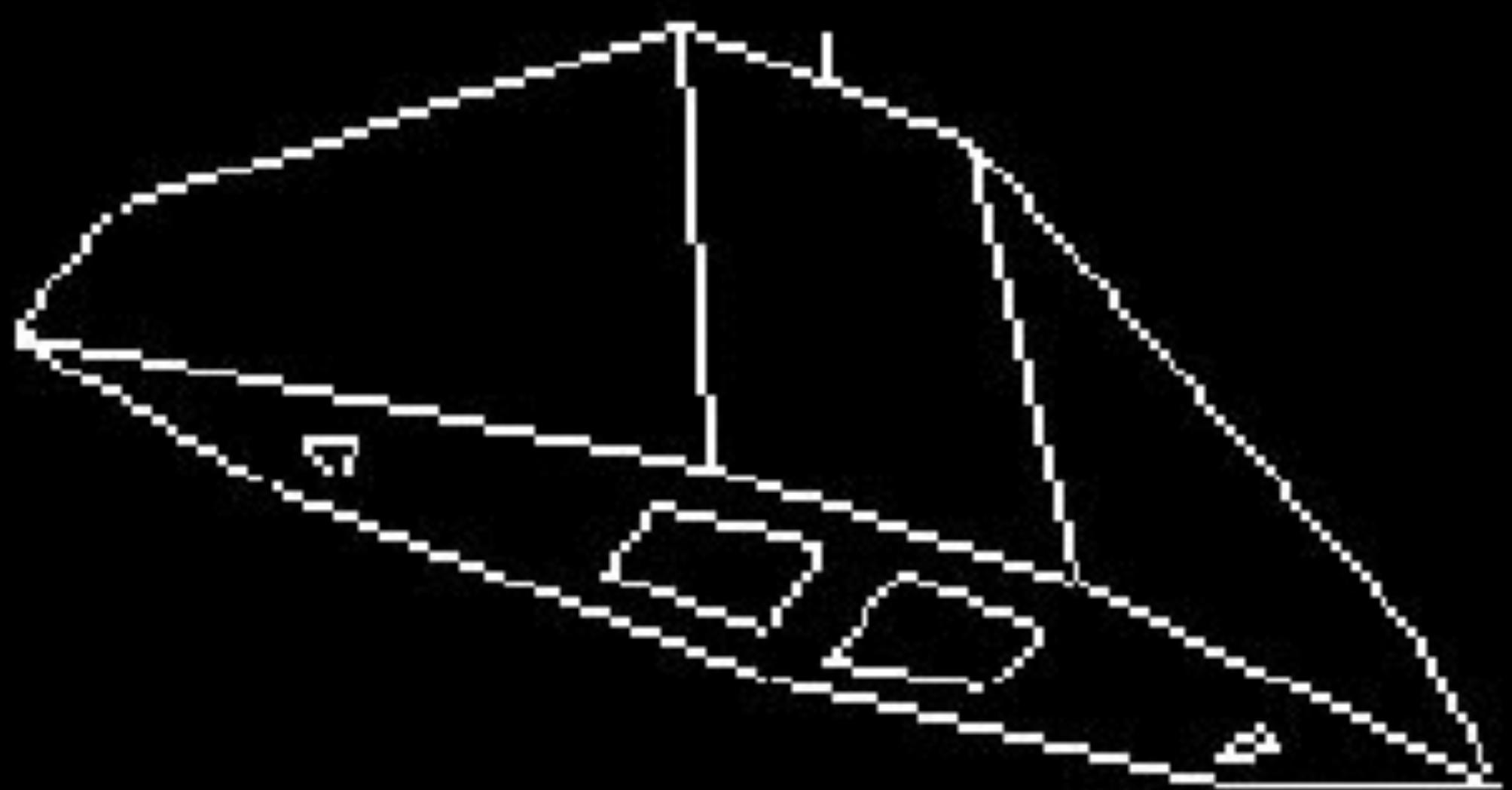
\$23C0	\$23C1	\$23C2	\$23C3	\$23C4	\$23C5	\$23C6	\$23C7
000000	000000	000000	000000	000000	000000	000000	000000
\$23C8	\$23C9	\$23CA	\$23CB	\$23CC	\$23CD	\$23CE	\$23CF
\$23D0	\$23D1	\$23D2	\$23D3	\$23D4	\$23D5	\$23D6	\$23D7
\$23D8	\$23D9	\$23DA	\$23DB	\$23DC	\$23DD	\$23DE	\$23DF
\$23E0	\$23E1	\$23E2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

KITTY  
000200



WORLD 1-1 TIME  
368





Load New Commander (Y/N)?

SPD: 0  
USI: 01.78  
THR:90%

HDG:0

ALT:83









The right side of the advertisement is a promotional page for the Voodoo2 graphics card. It features a large, stylized eye of a voodoo doll looking intensely at the viewer. Below the eye, the words "Voodoo2" are written in a large, metallic, textured font. A banner across the middle of the page reads "Get the Magic of Speed!"

**Awards:**

- Hardware AWARD PC-Games 05/98
- Hardware AWARD PC-Aktion 05/98
- PC AWARD PC-Player 05/98

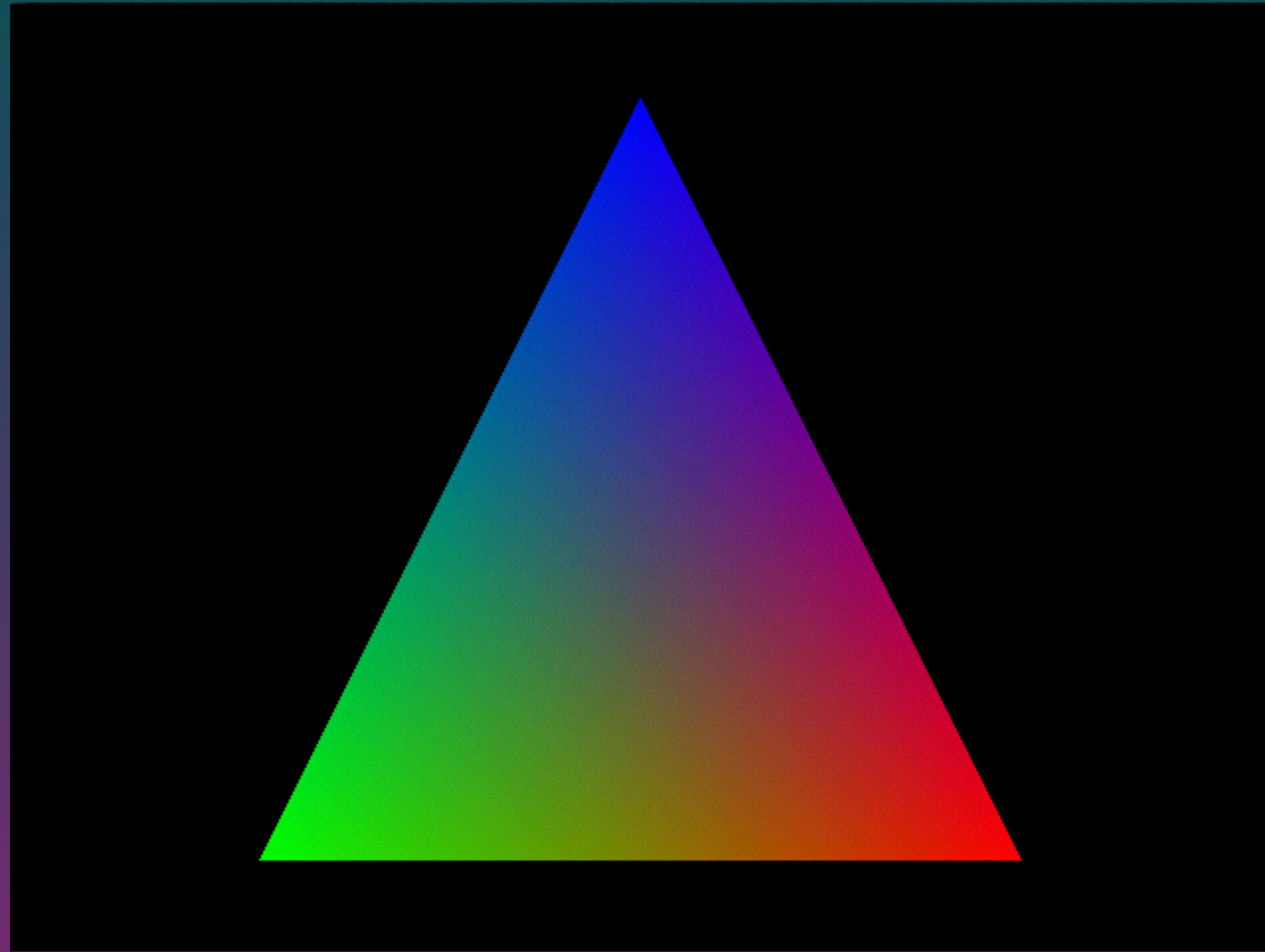
**4 Faszinierende Spiele**

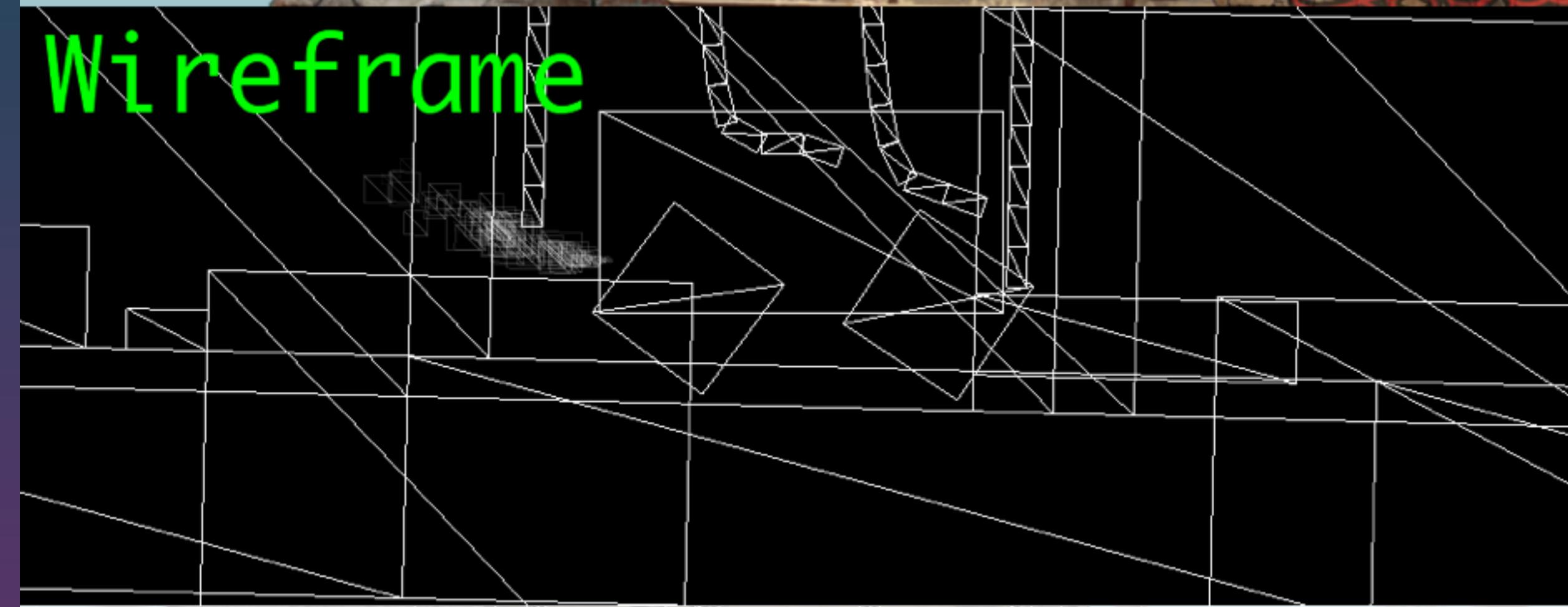
The page also shows four small game demos: "SP Chopper", "Incoming", "actua SOCCER 2", and "ULTIMATE RACE PRO".

**Features:**

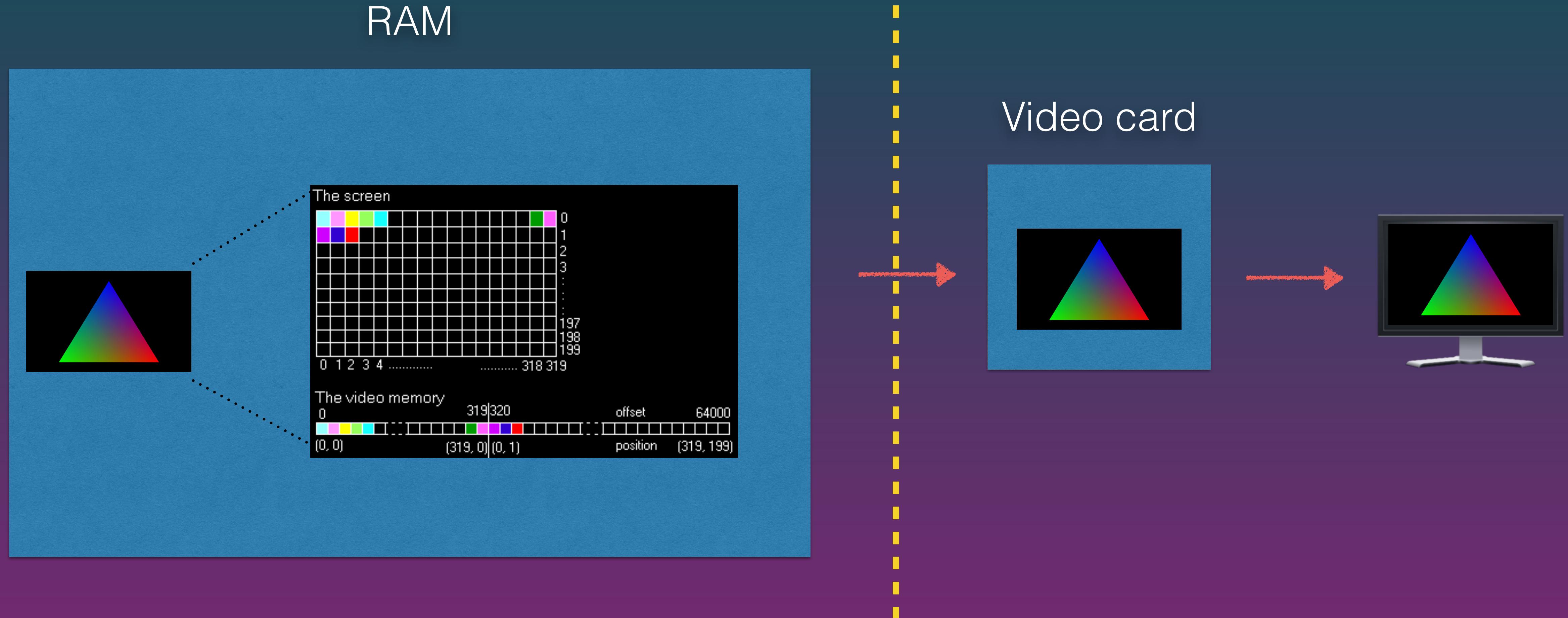
- Basiert auf dem neuen Voodoo2 Graphics™-Chipsatz von 3Dfx Interactive™
- Ausgestattet mit 12 MB Hochleistungsspeicher für stärkste Leistung
- Bis zu 50 Milliarden Operationen und 3 Millionen Triangles pro Sekunde
- Bis zu dreimal schnellere 3D-Verarbeitung als beim ursprünglichen Voodoo Graphics™-Chipsatz!
- Arbeitet mit Ihrer vorhandenen Grafikkarte zusammen und bietet Ihnen das schnellste 3D-Spiel aller Zeiten
- Enthält 4 topaktuelle Spiele







# Software Rendering



# Software Rendering



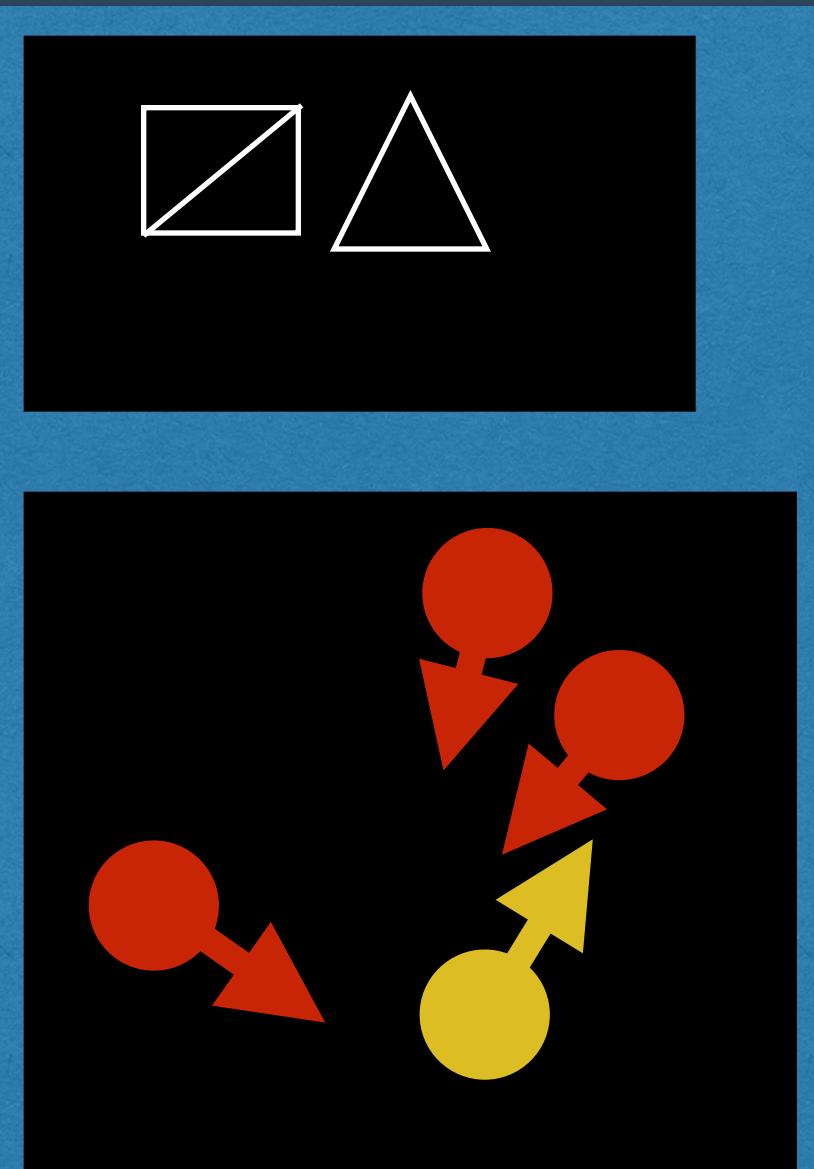
$2560 * 1600 = 4,096,000$  pixels

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

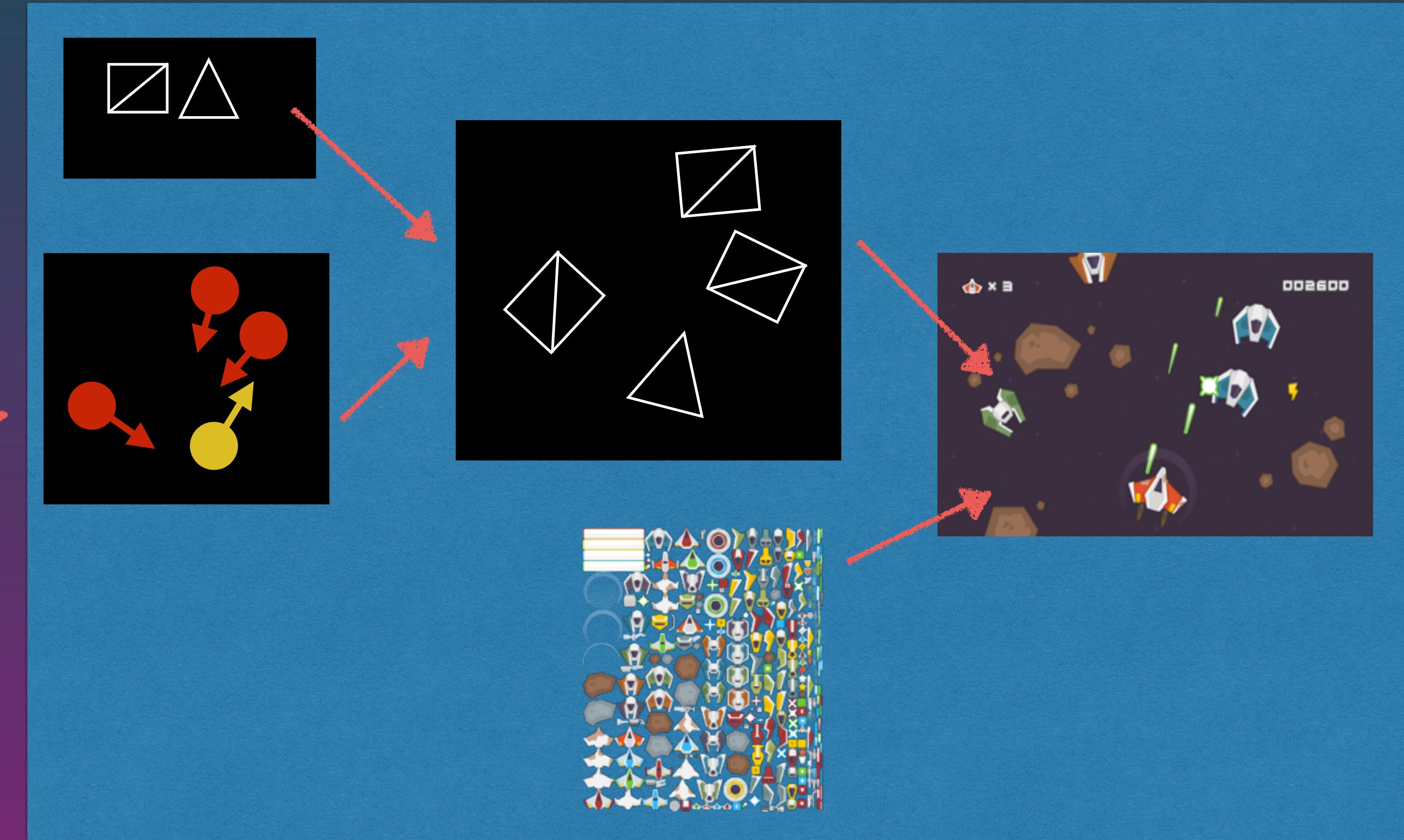
60 times a second

# Hardware Rendering

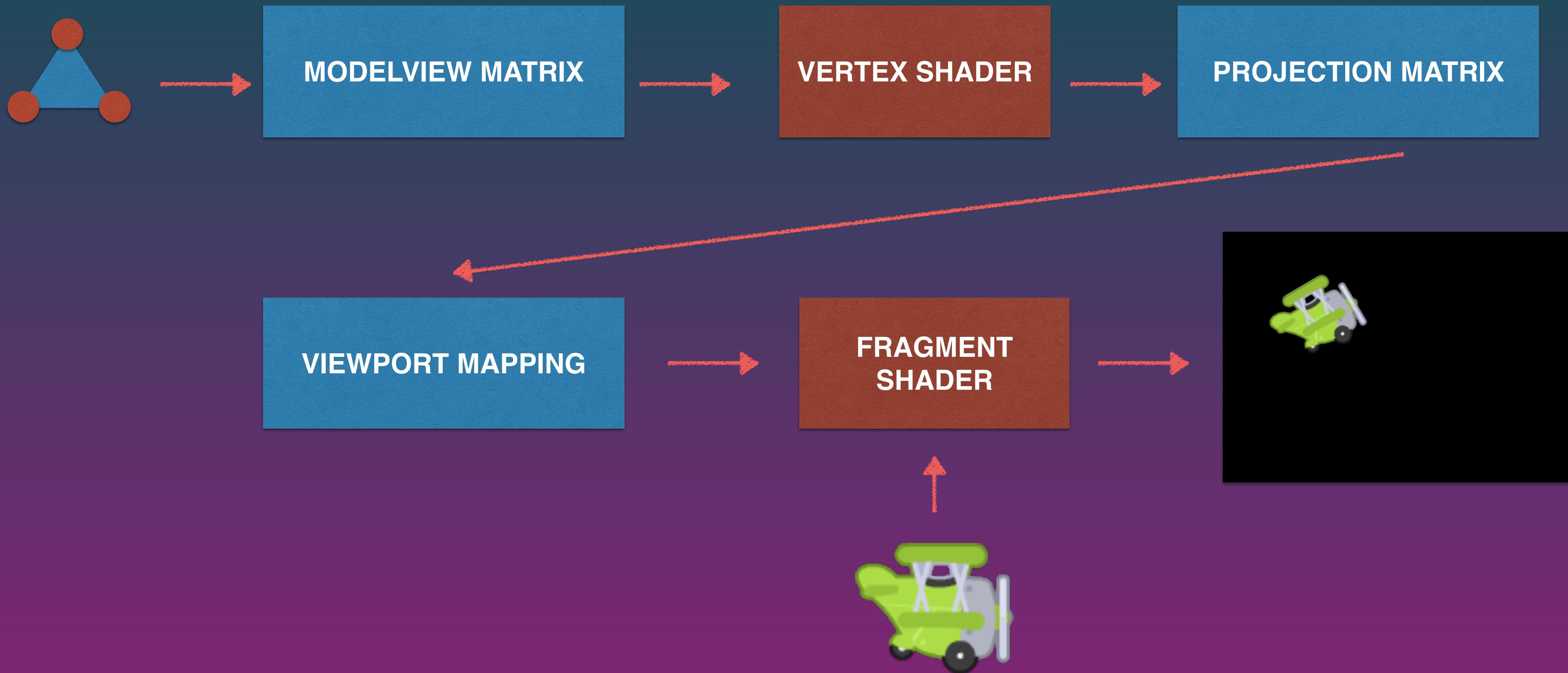
RAM



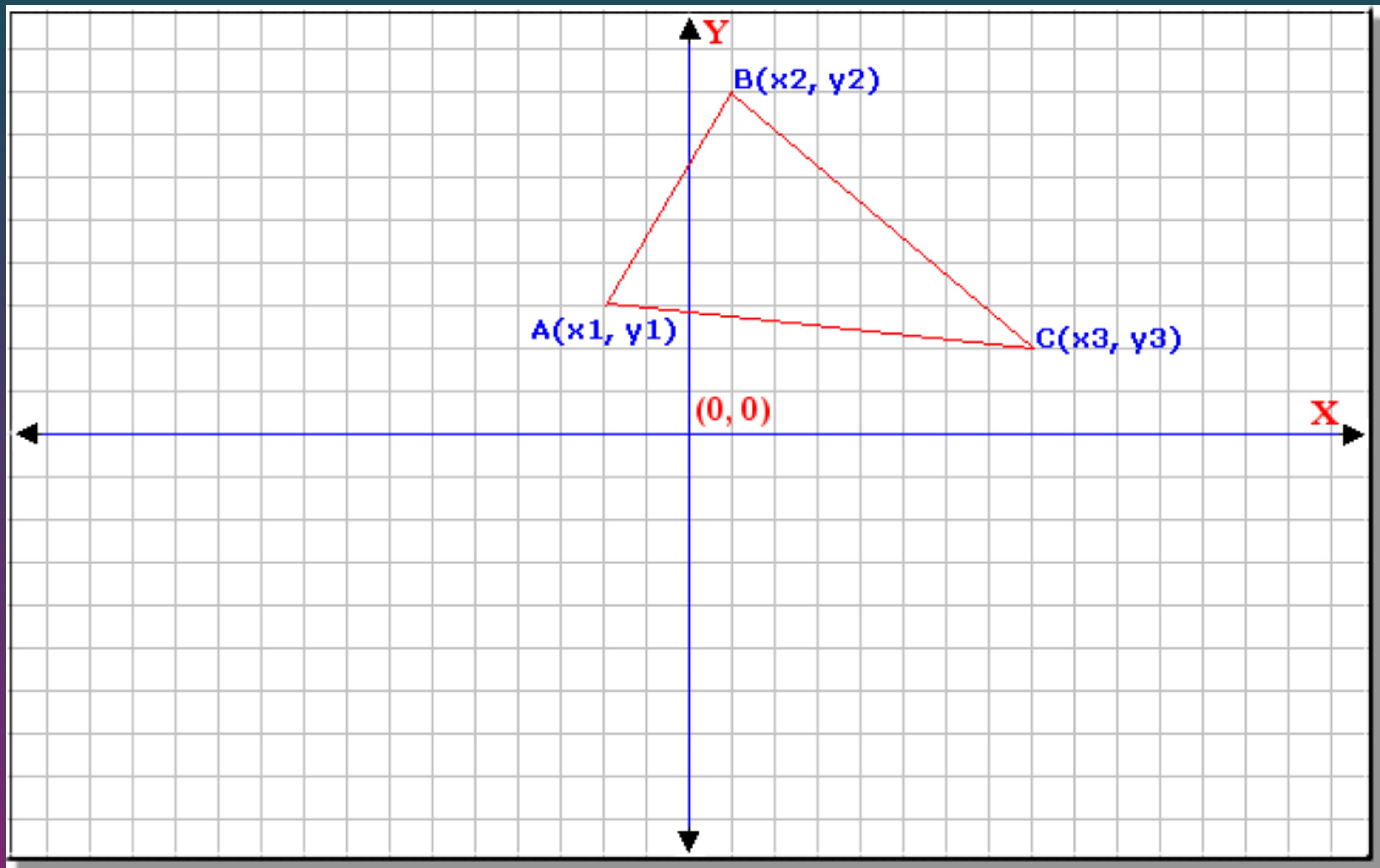
Video card



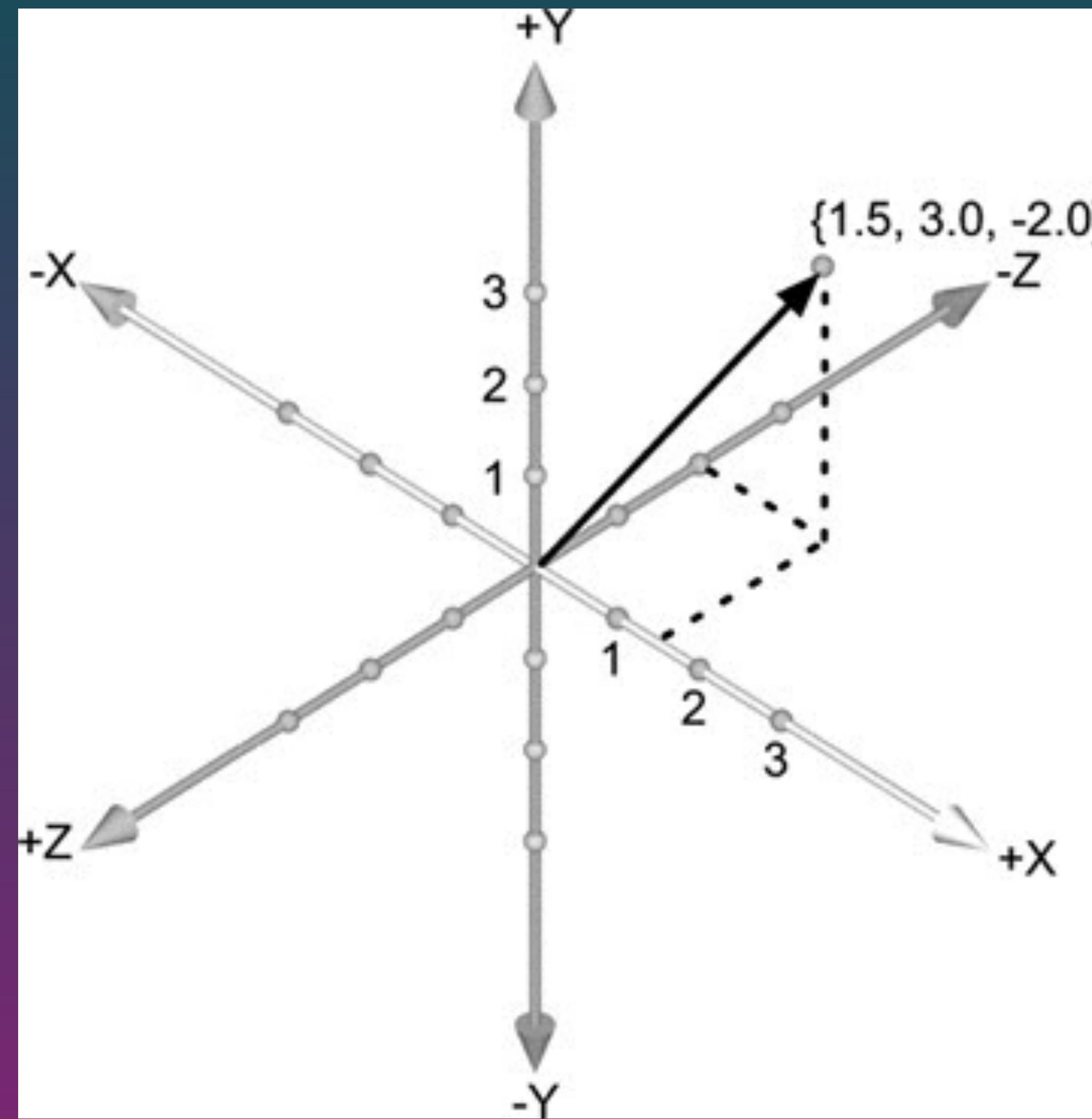
# The rendering pipeline.

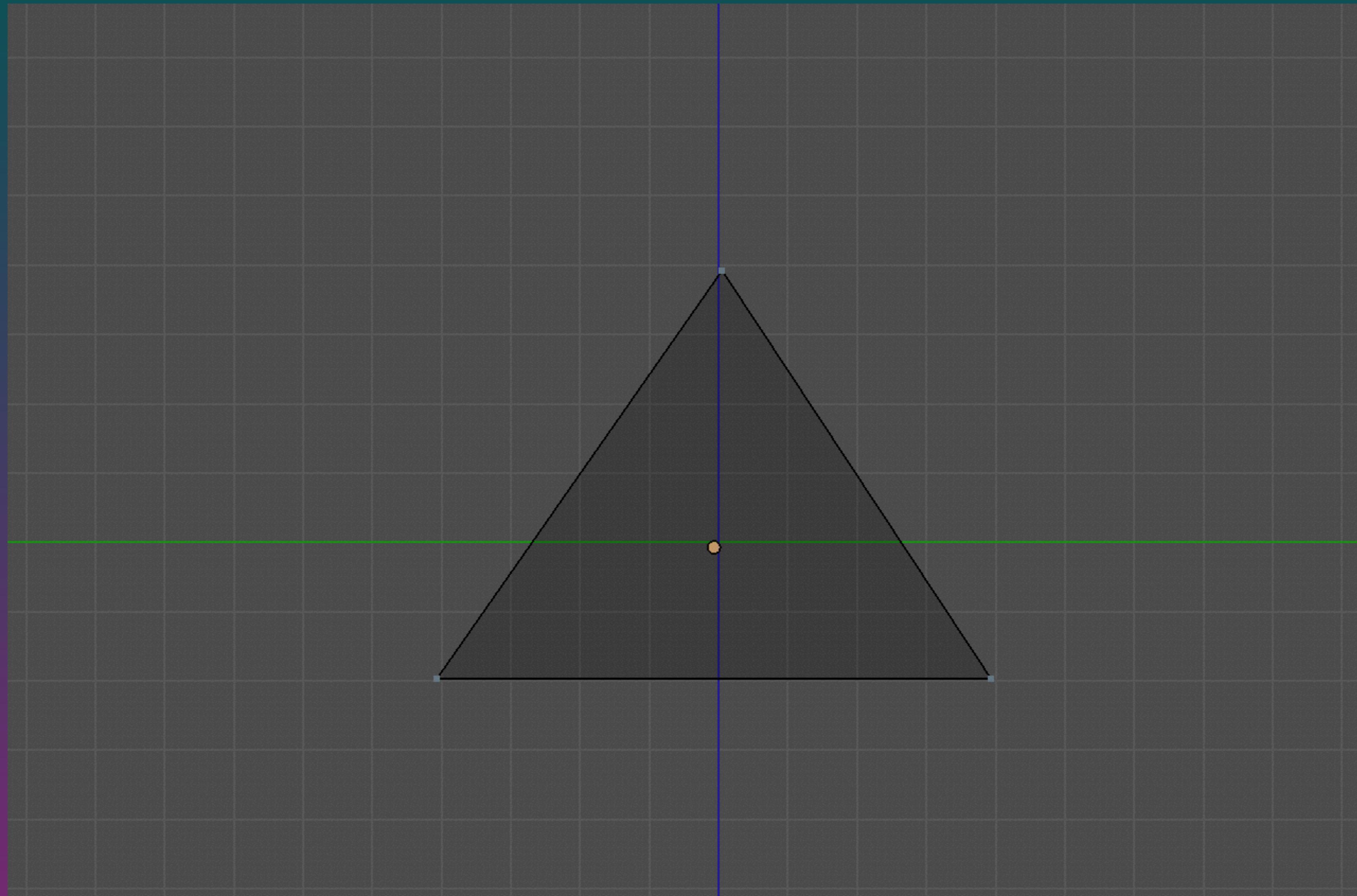


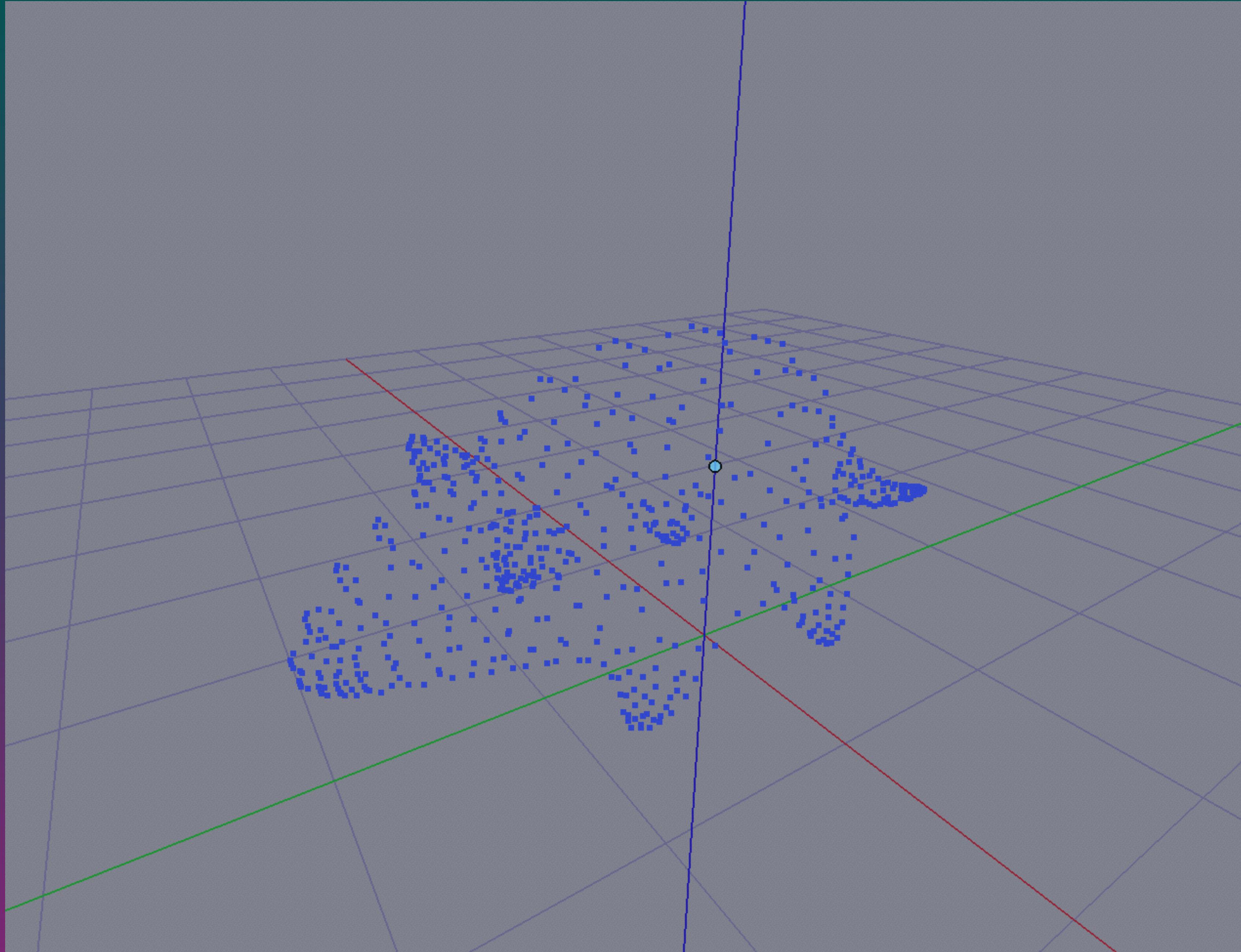
# Vertex data

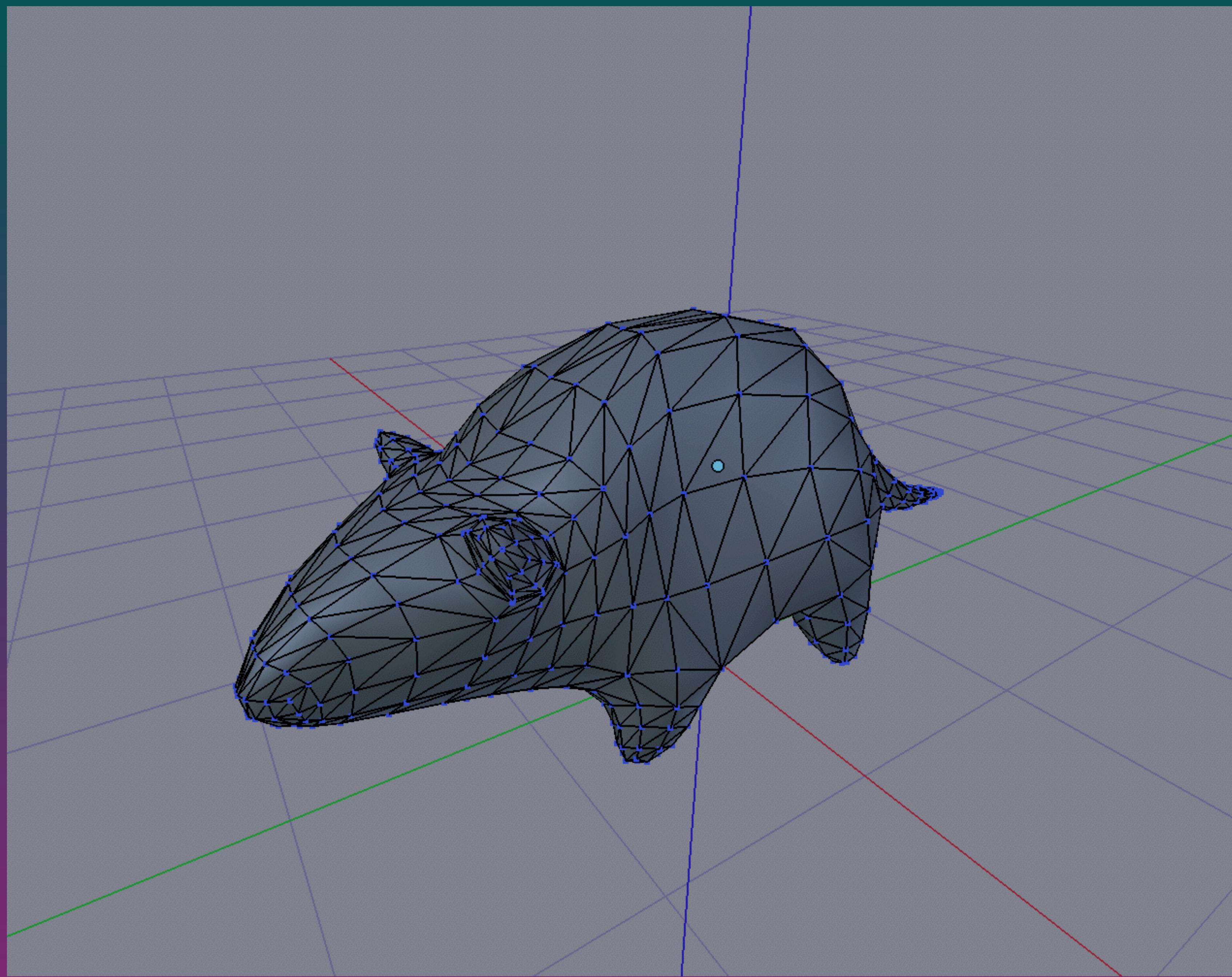


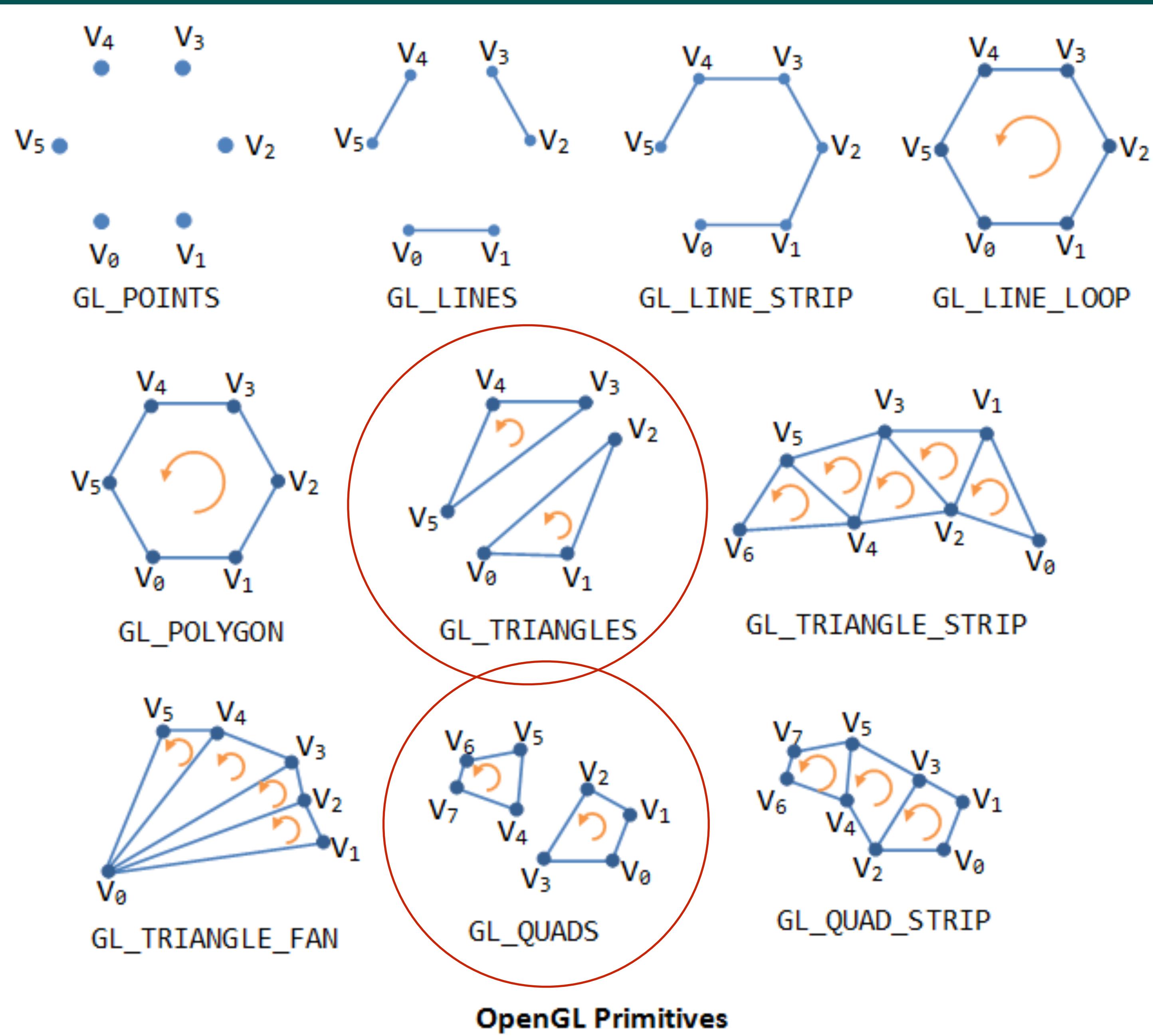
# Vertices



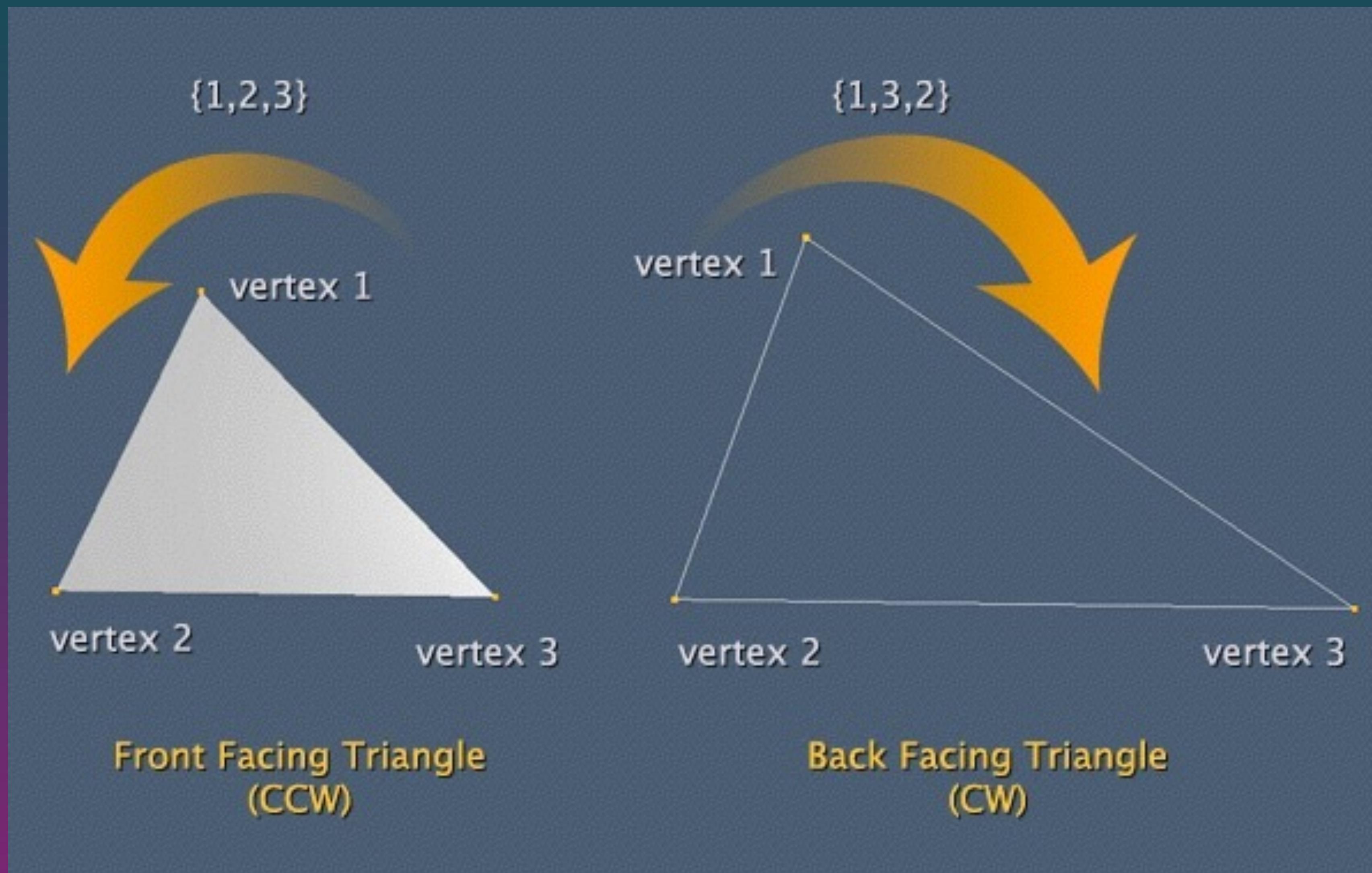


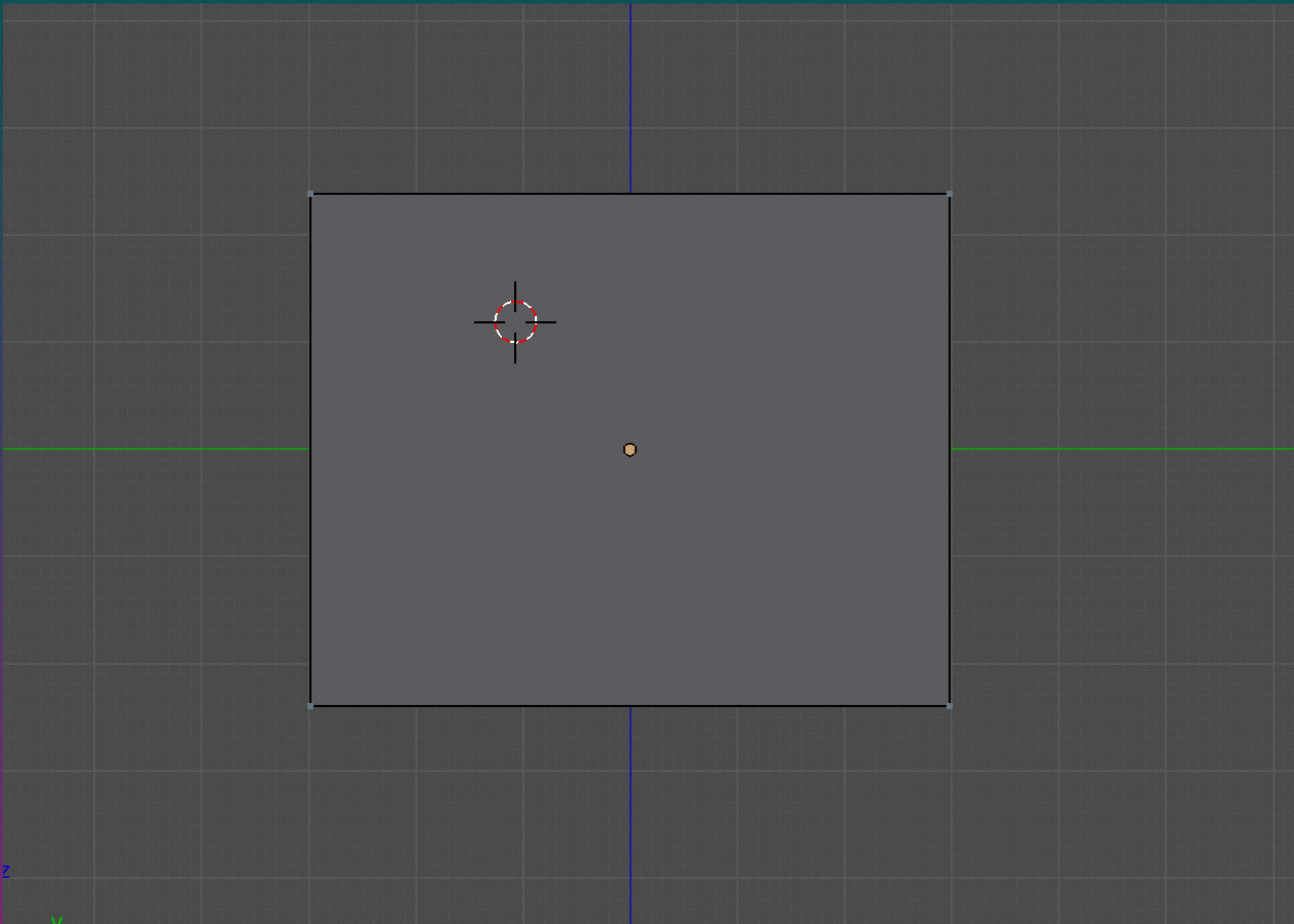






Polygons are one-sided and the side is defined by the order of the vertices.







Z

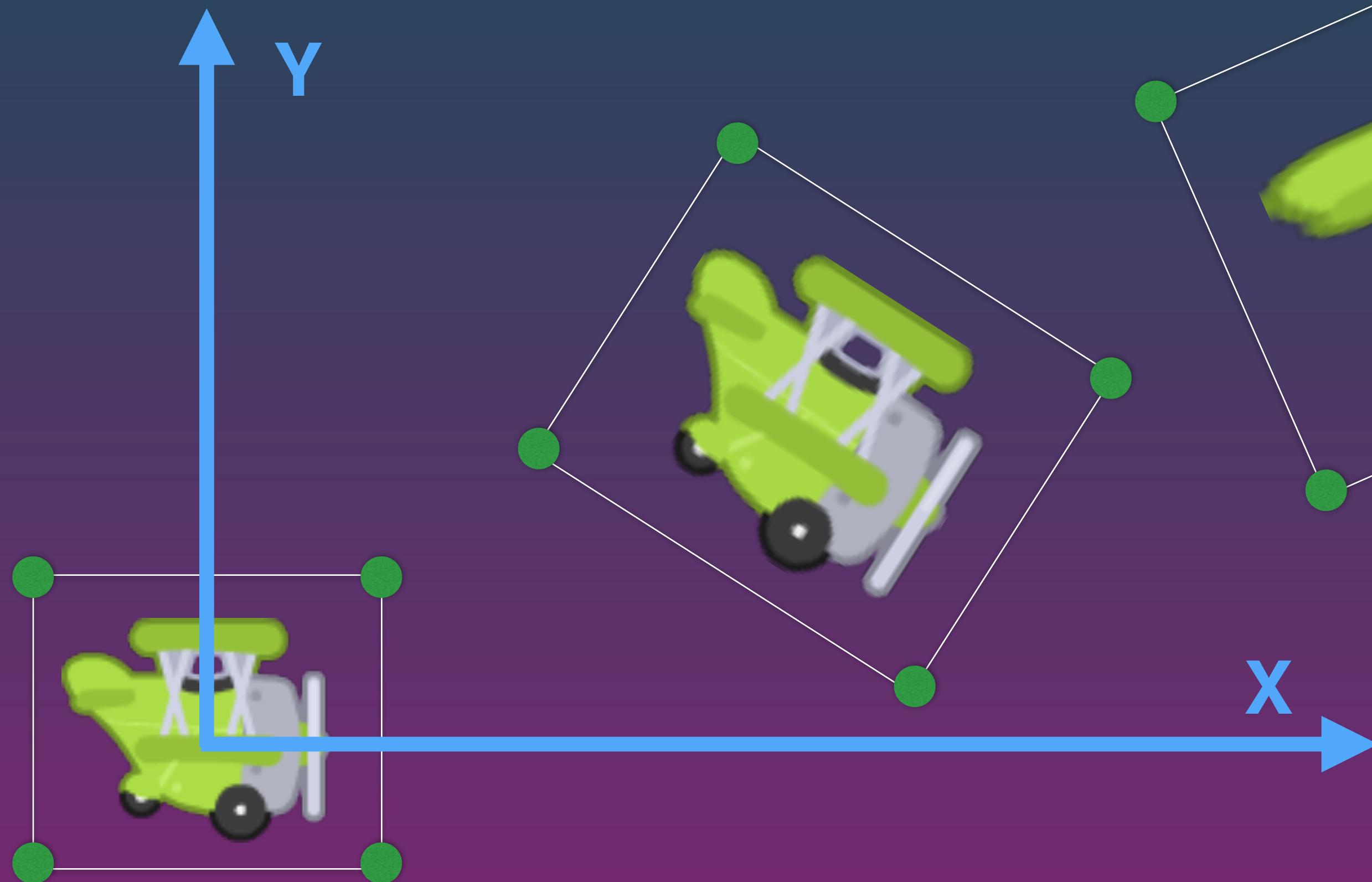
# Modelview matrix.

# 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

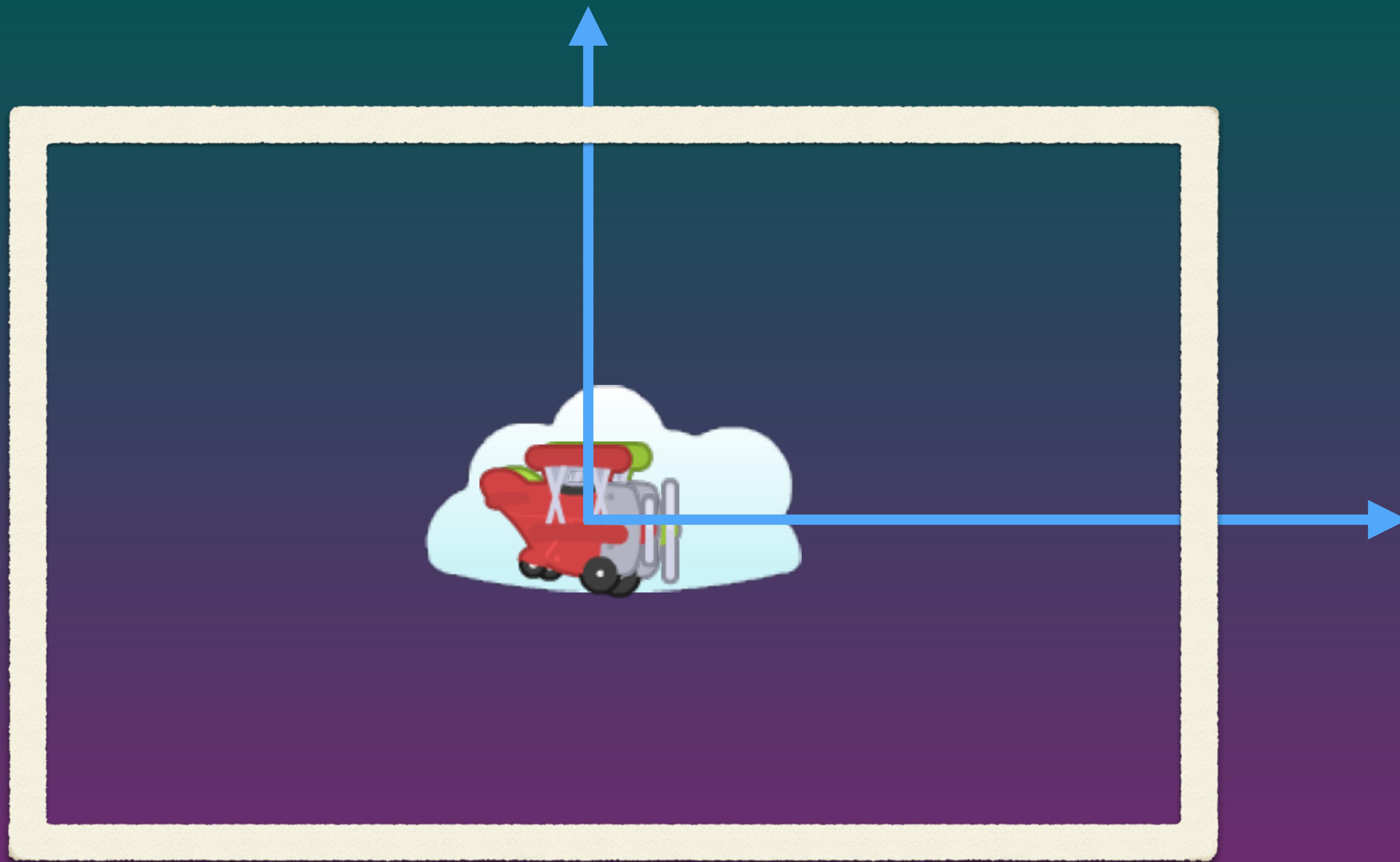
# Modelview matrix

=

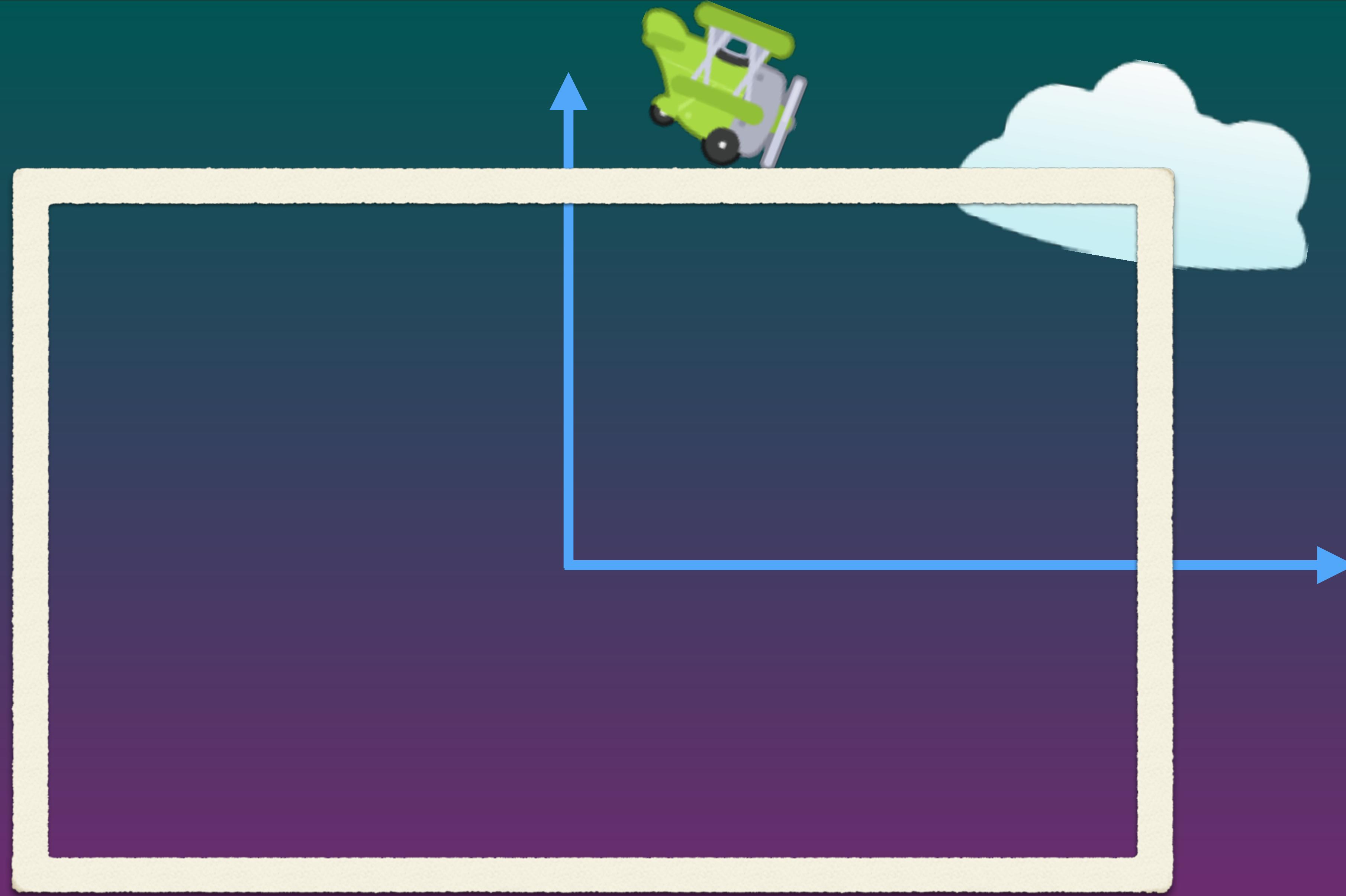
Model transformation matrix

\*

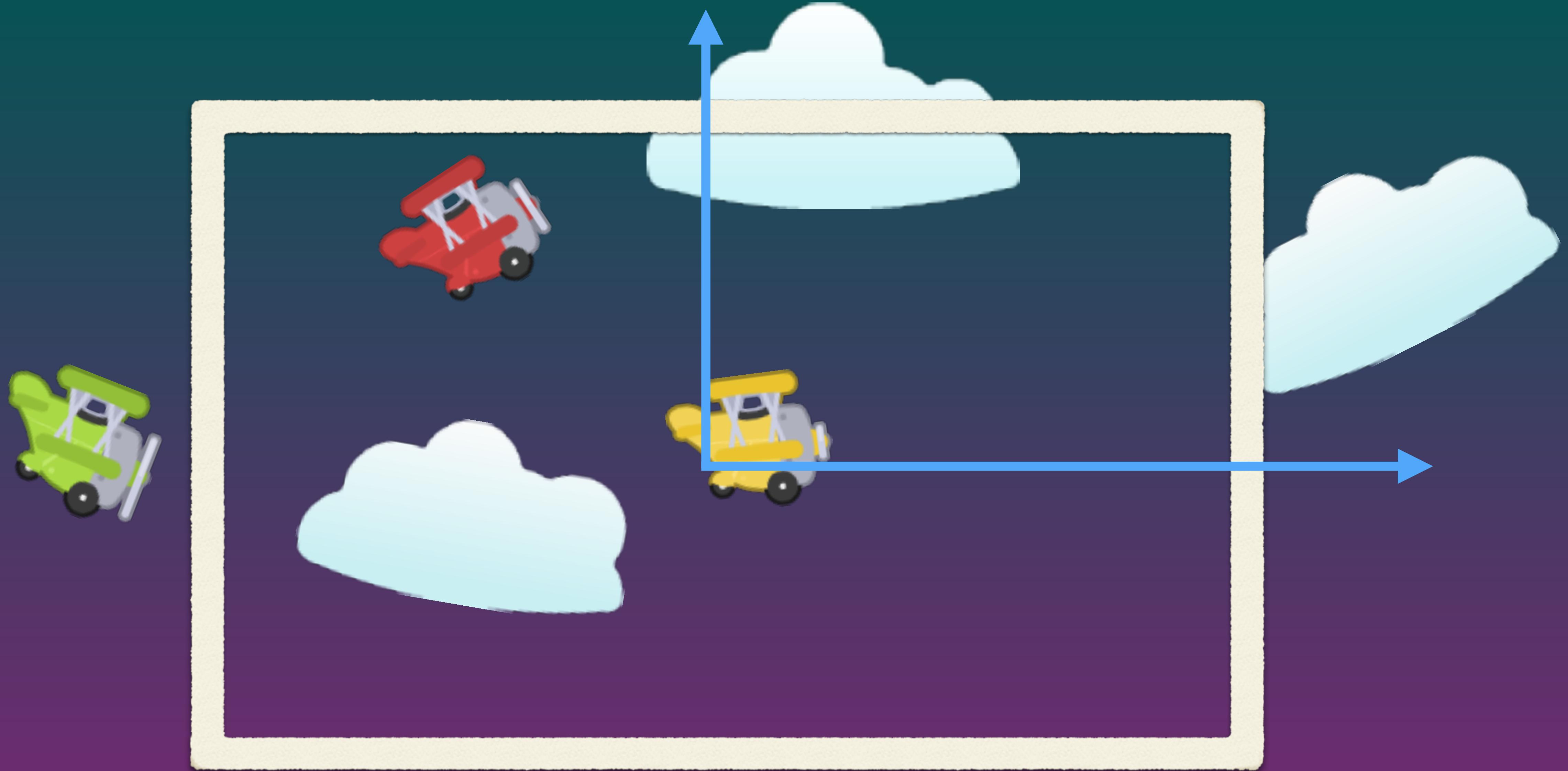
View transformation matrix.



Without model or view matrices :(



Without view matrix

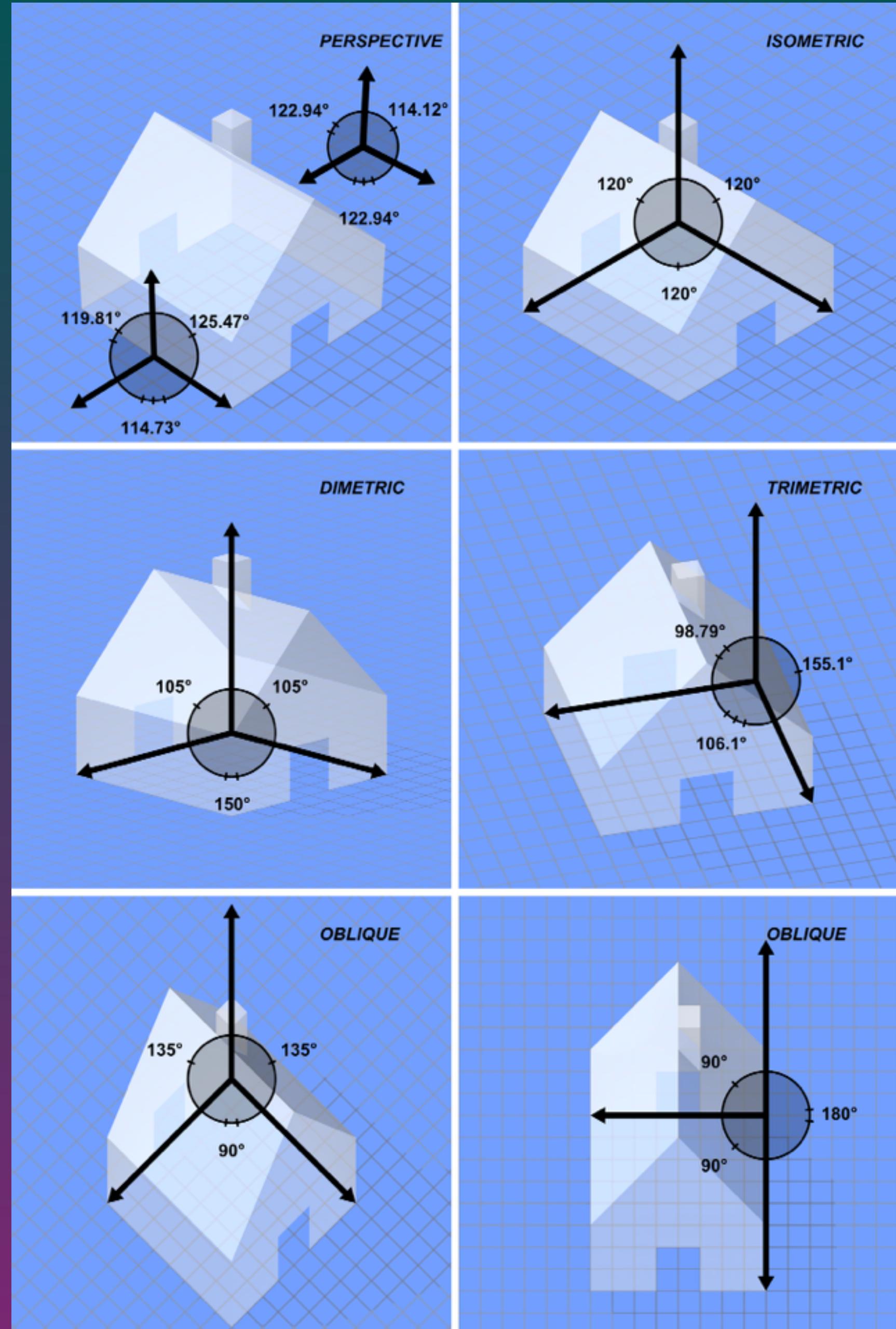


The view matrix is the **INVERSE** of our camera's transformation matrix

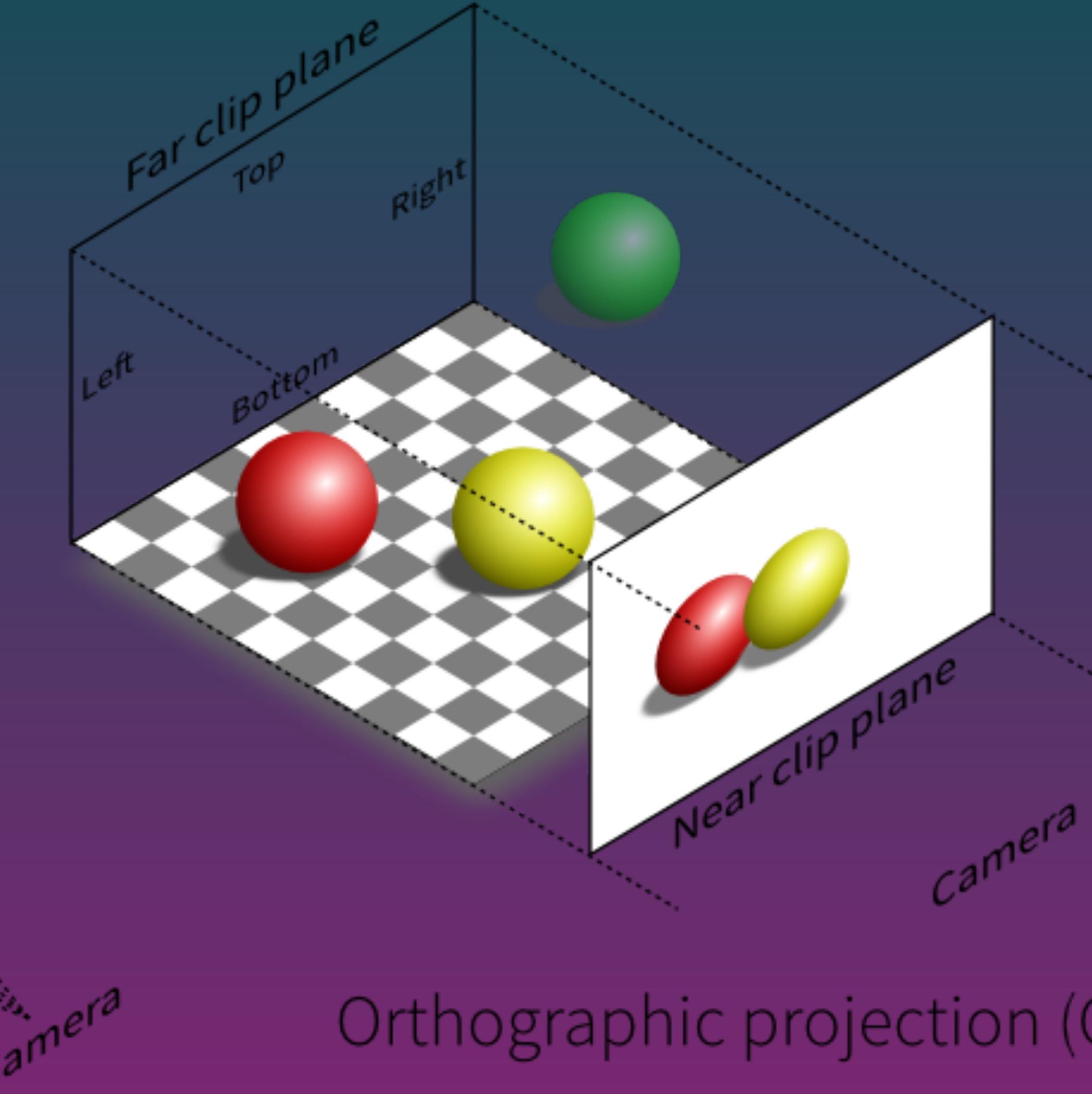
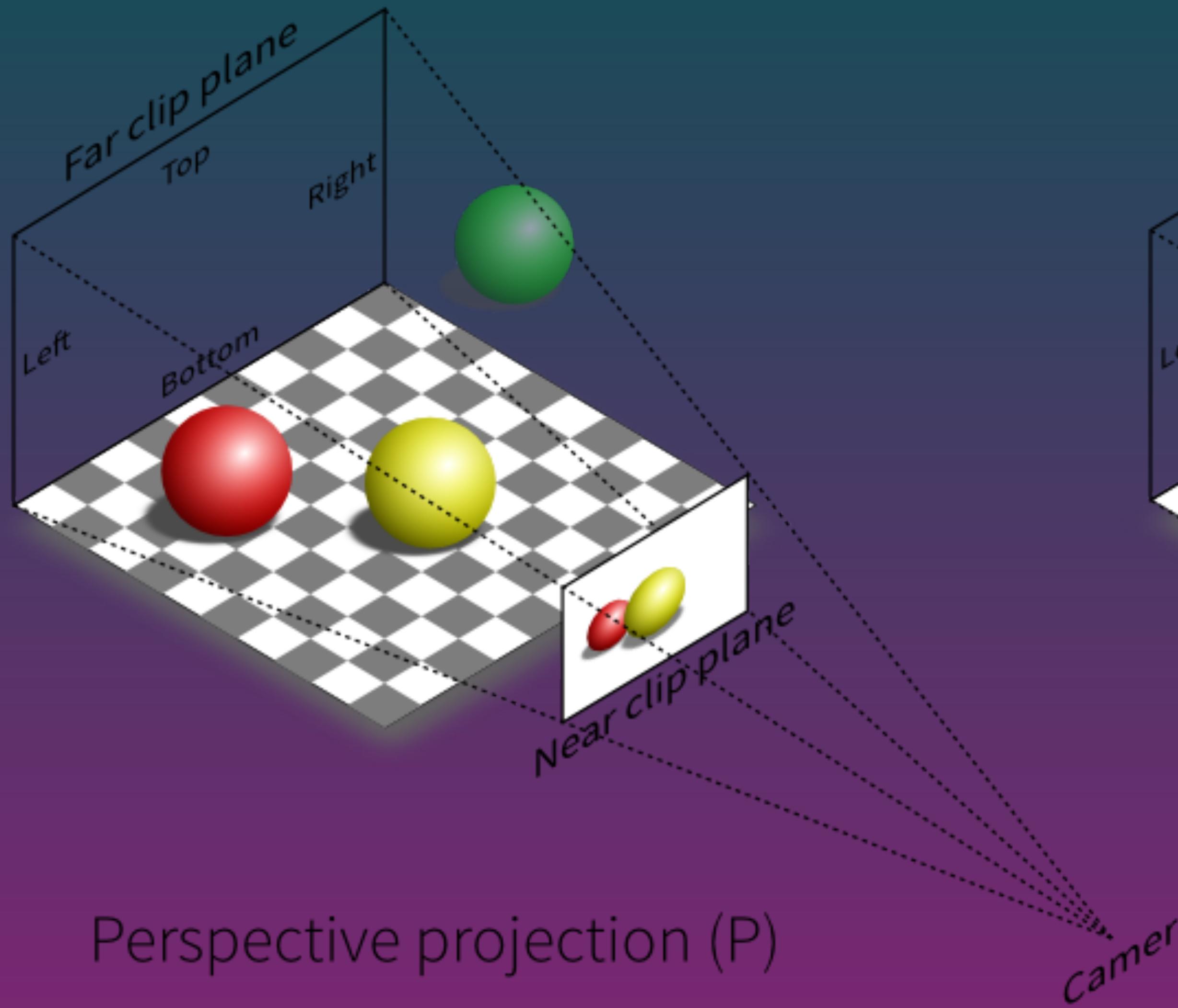
# Projection matrix.

# What is projection?

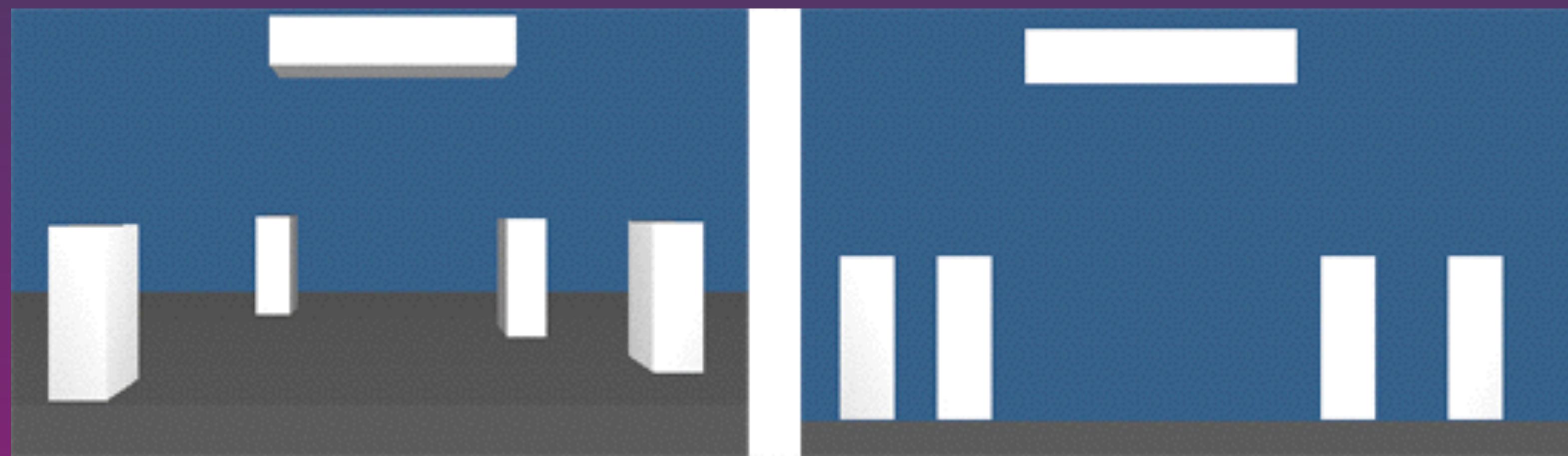
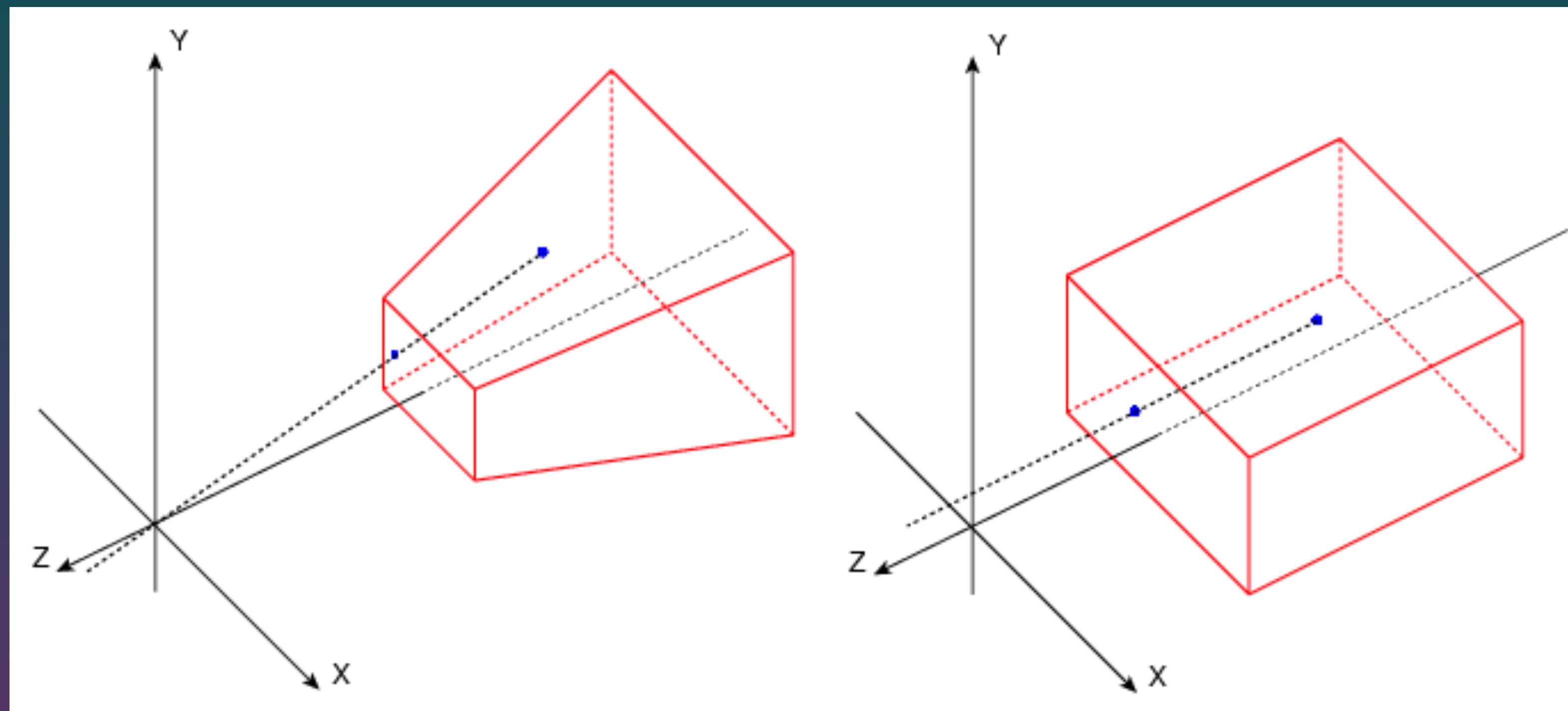
A means of representing a three-dimensional object in two-dimensions.



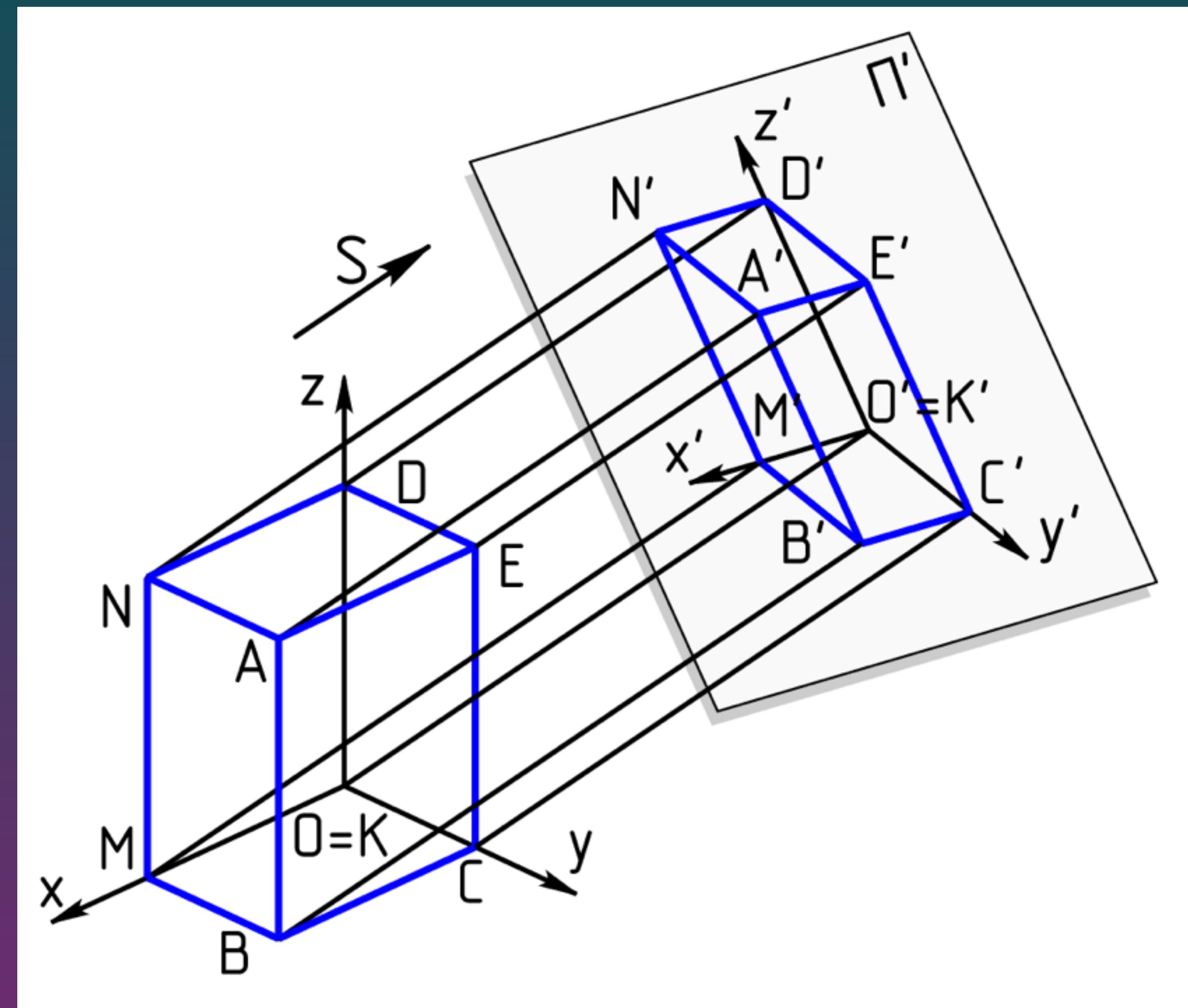
# Parallel vs. Orthographic projection.



# Parallel vs. Orthographic projection.



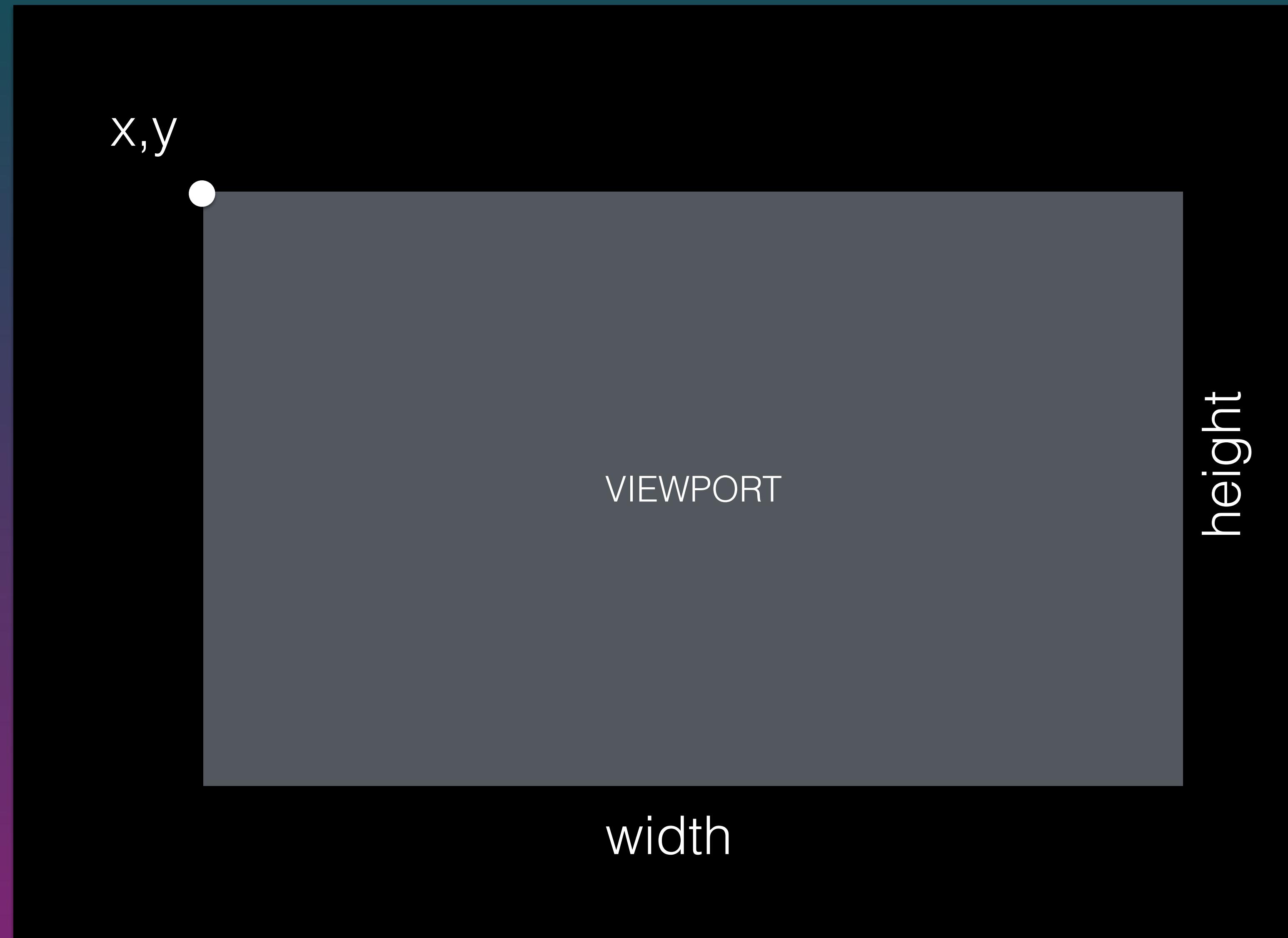
# What is a projection matrix?



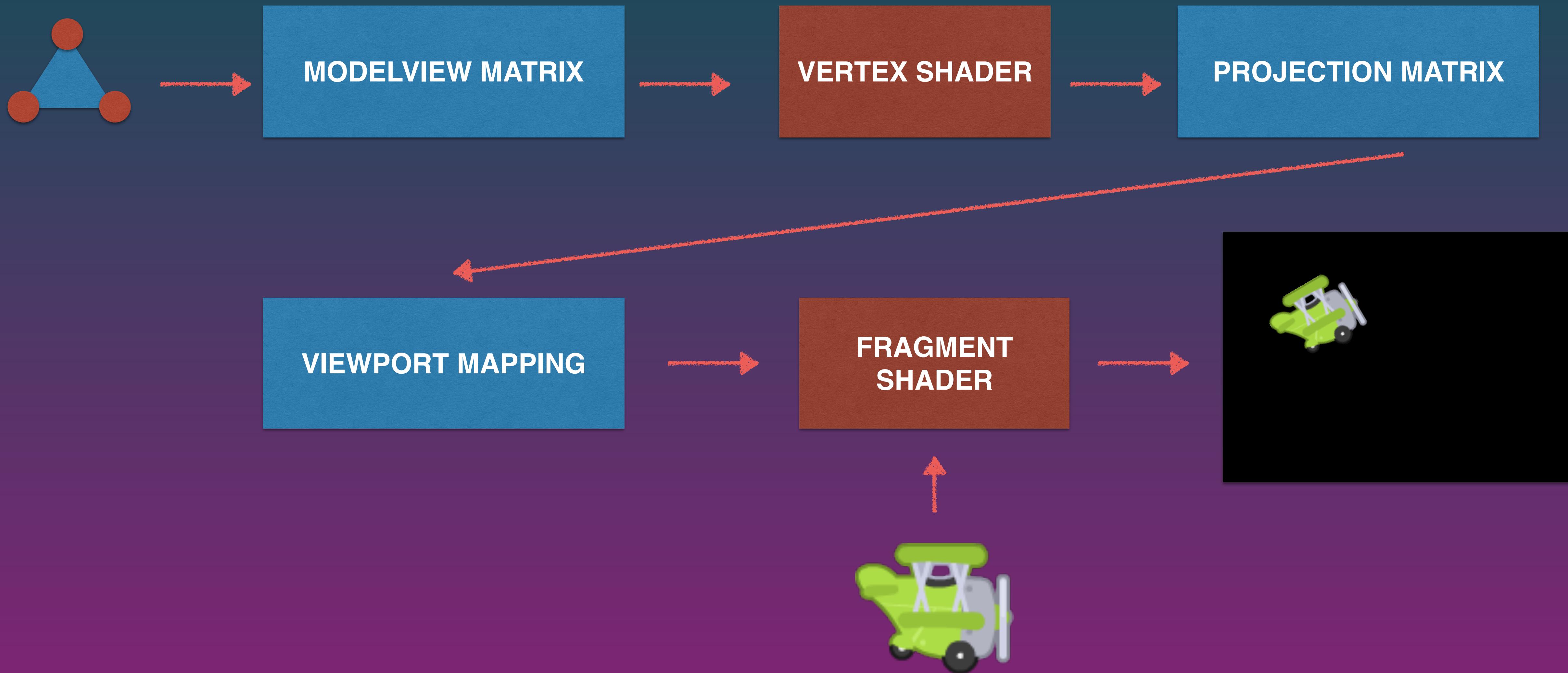
Vertex position \* projection matrix = screen vertex position

Viewport mapping.

# Offset and size of rendering area in pixels.



# The rendering pipeline.



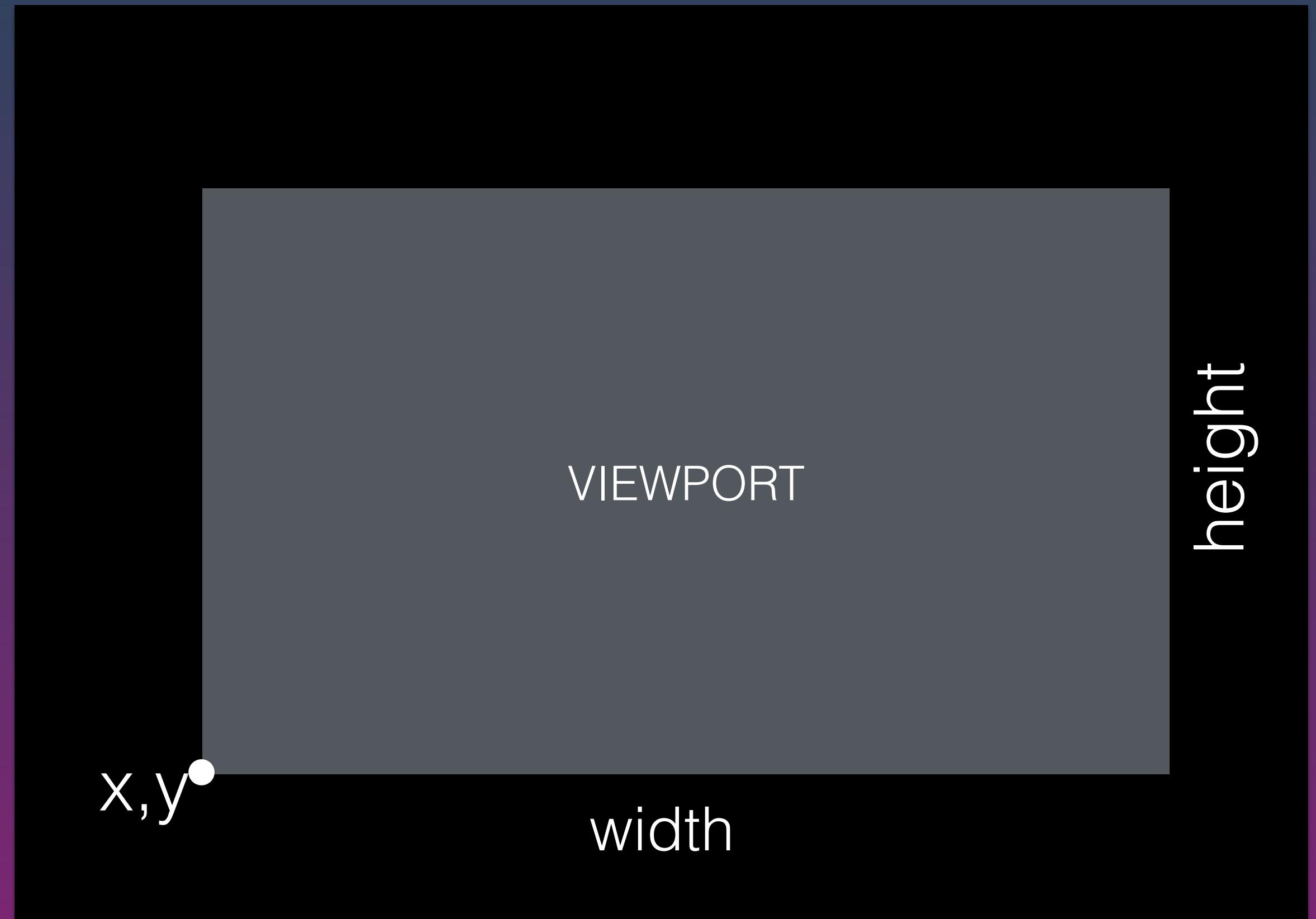


The setup.

```
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.

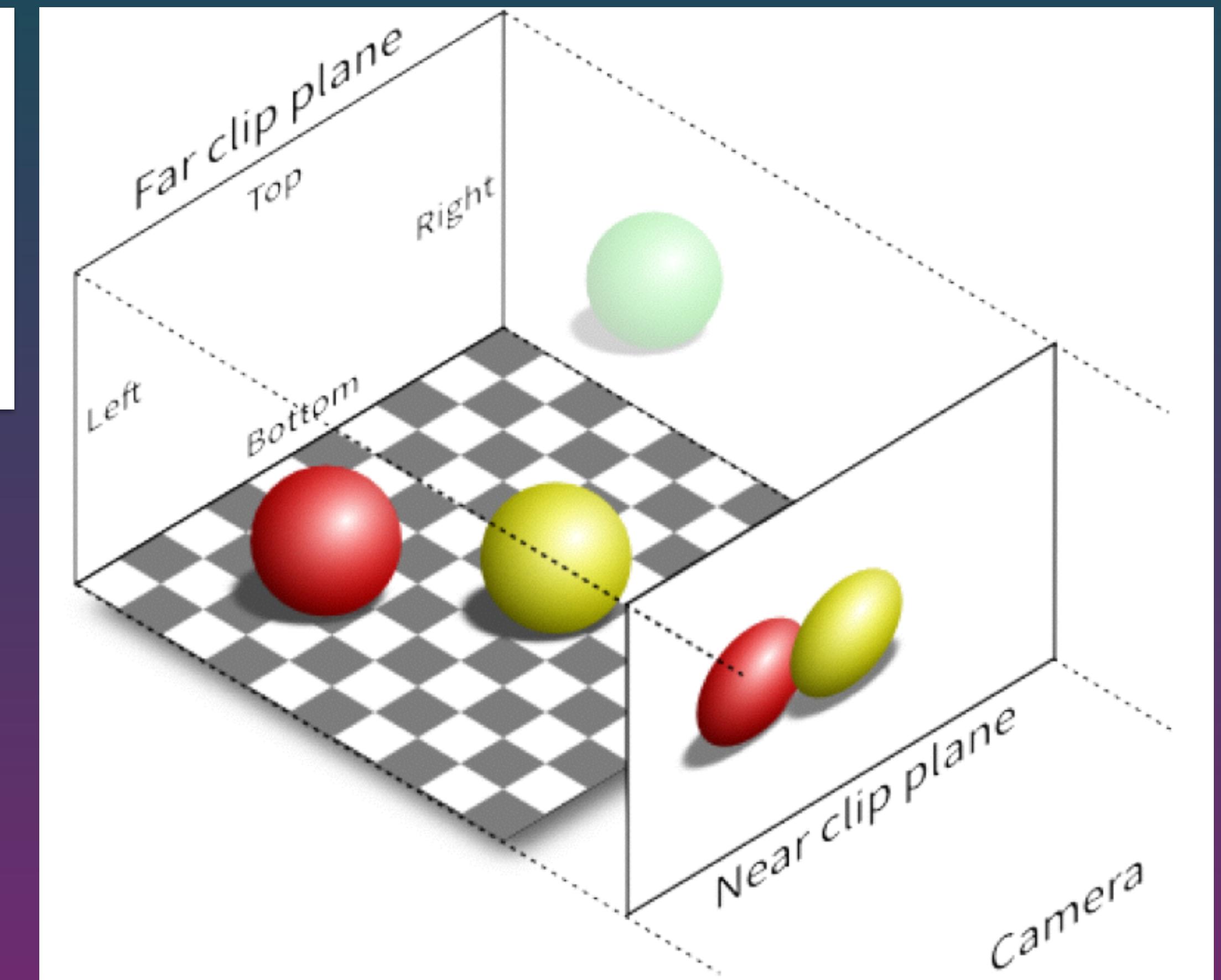
```
glMatrixMode(GL_MODELVIEW);
```

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);
```

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

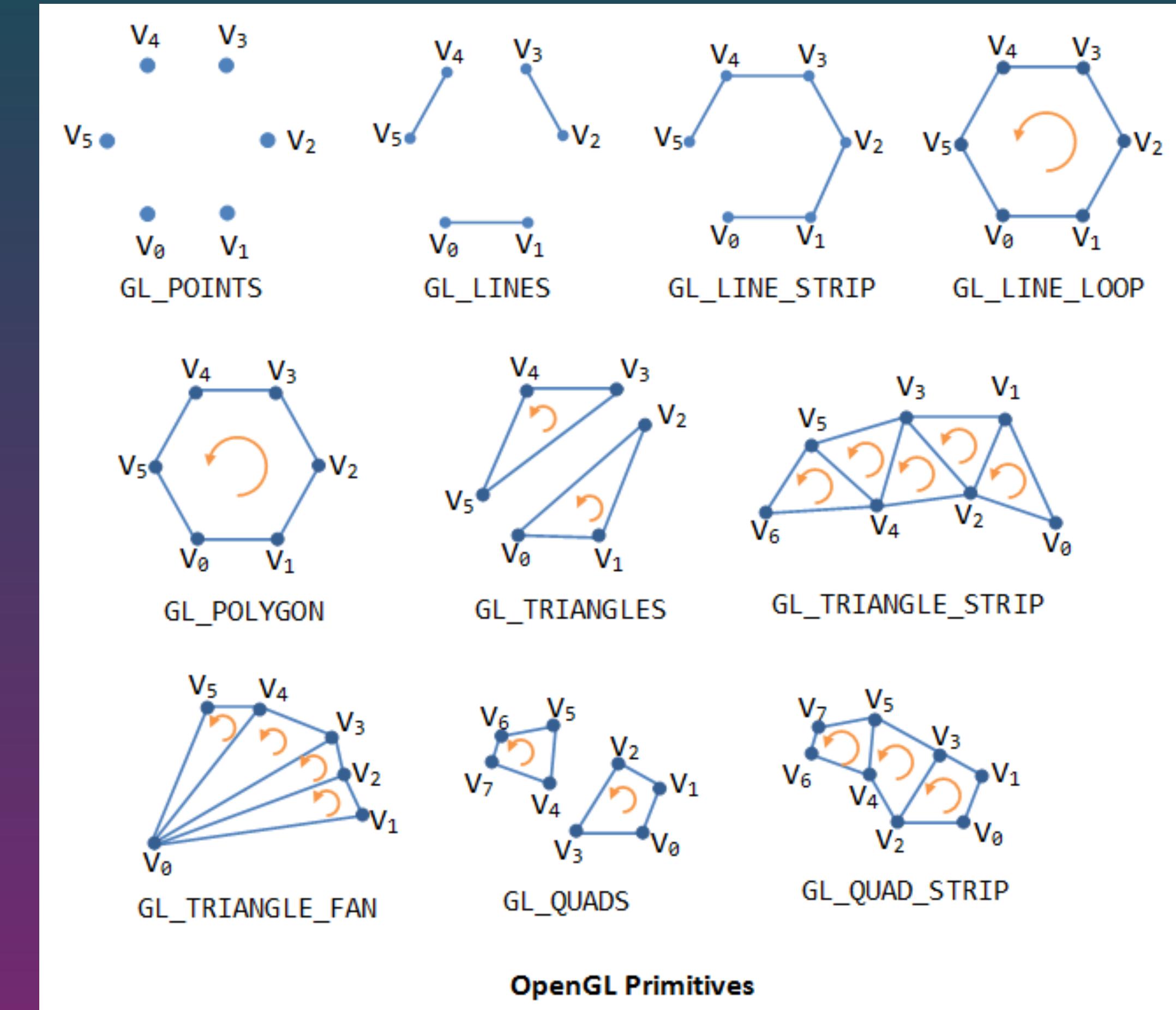
Enables a client state capability.

```
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).

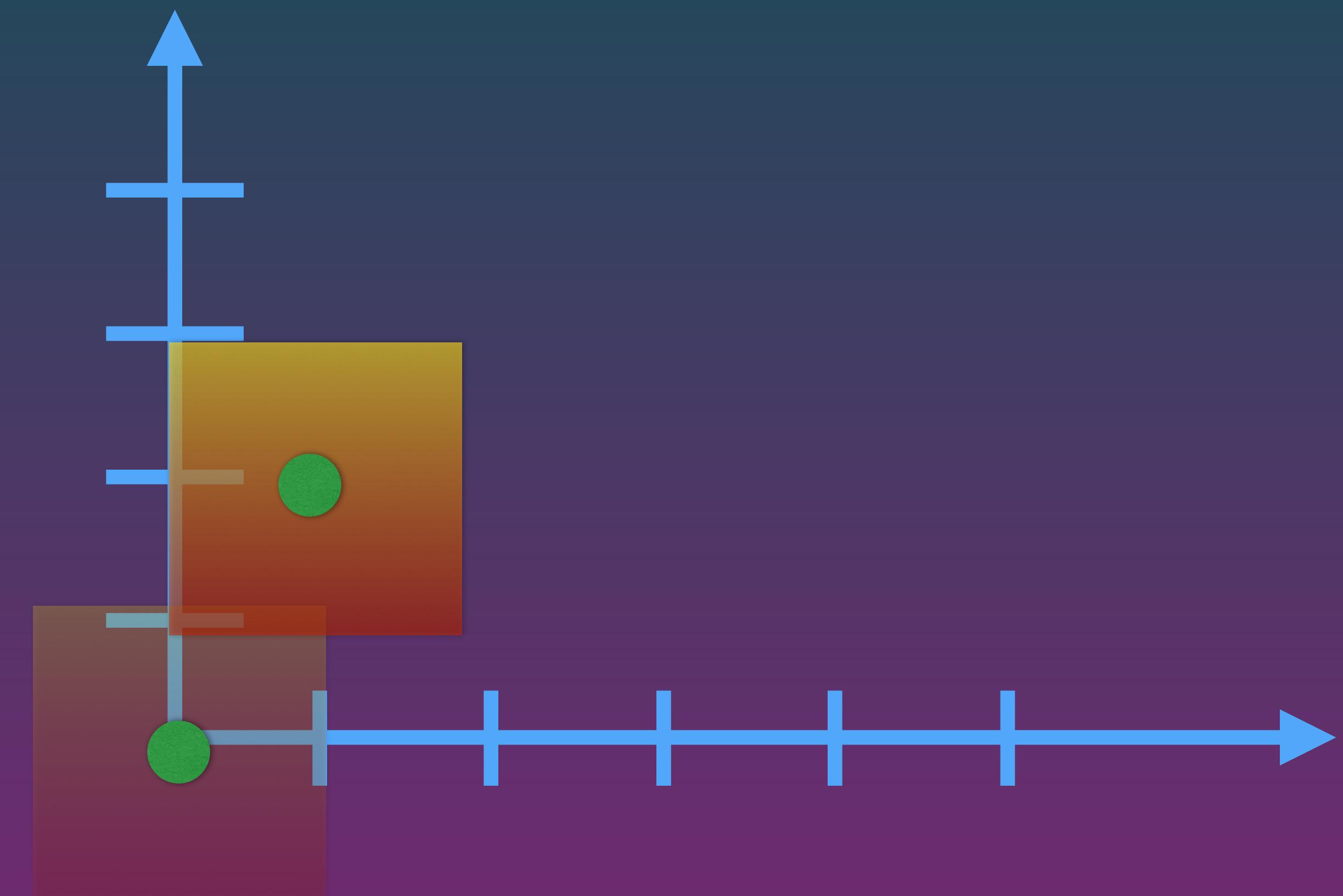
```
void glScalef (GLfloat x, GLfloat y, GLfloat z);
```

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).

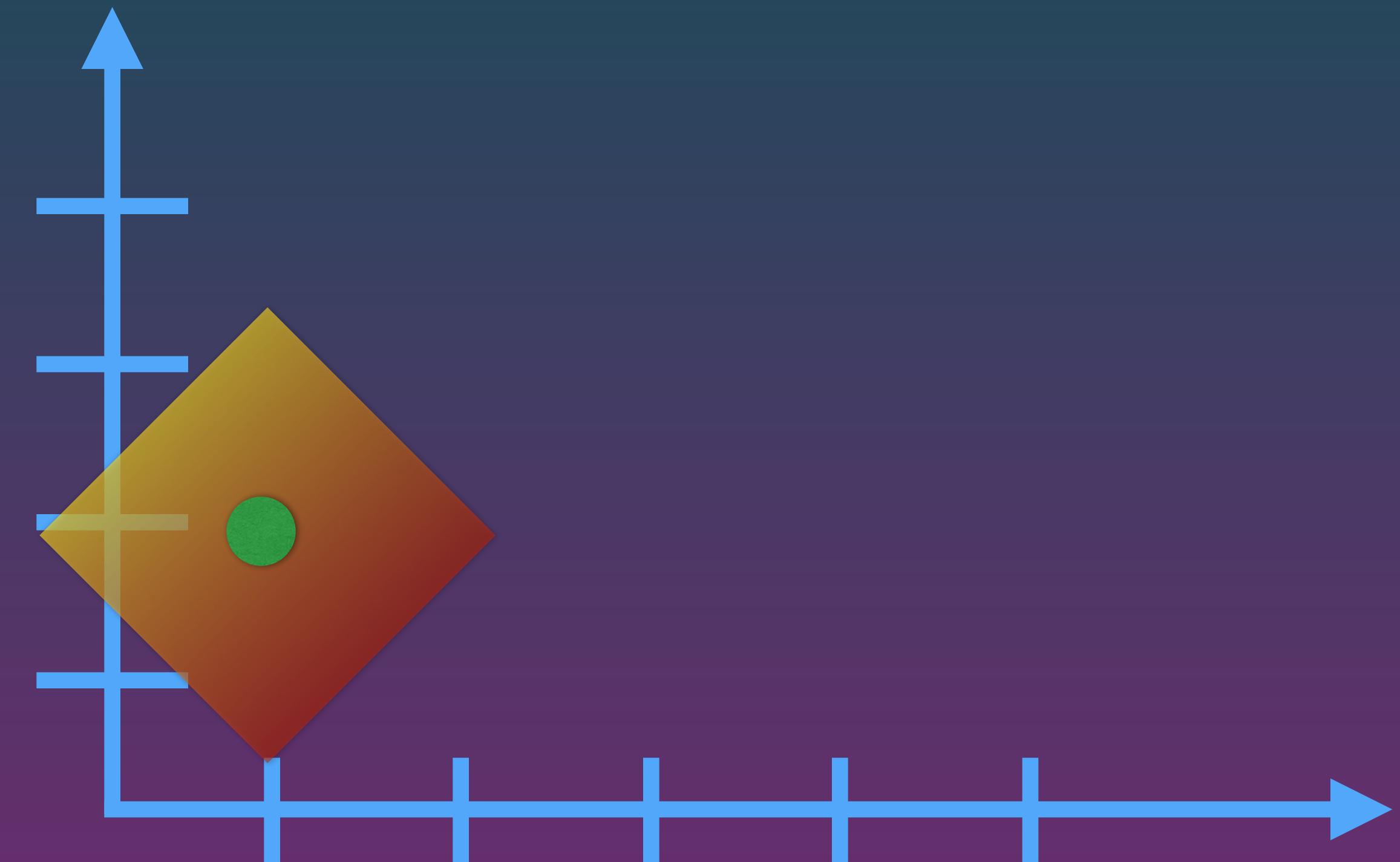
```
glLoadIdentity();  
glTranslatef(1.0, 2.0, 0.0);
```



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

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

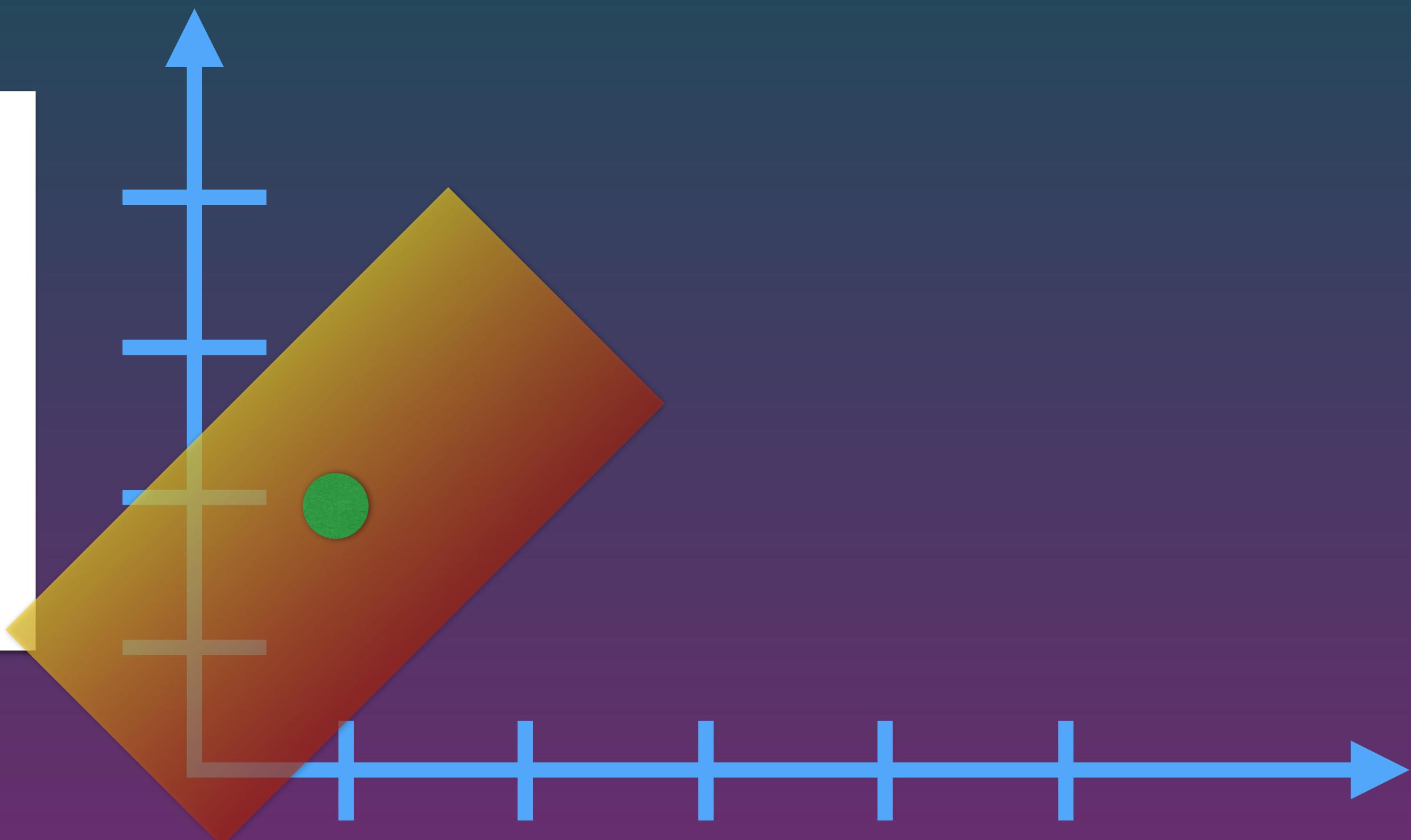
```
glLoadIdentity();  
glTranslatef(1.0, 2.0, 0.0);  
glRotatef(45.0f, 0.0, 0.0, 1.0);
```



```
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.