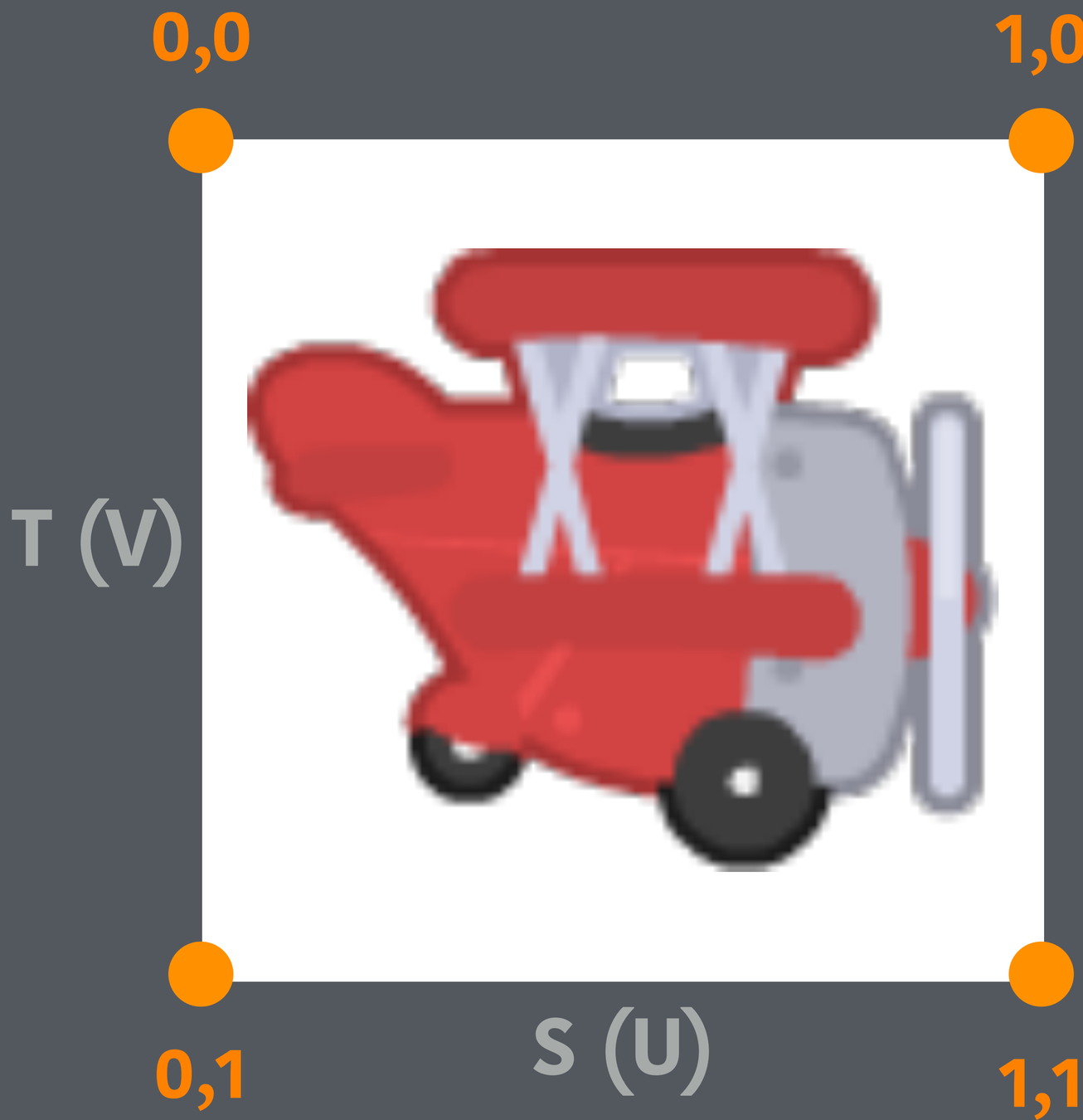
Graphics Foundations

Part 3

Texture coordinates

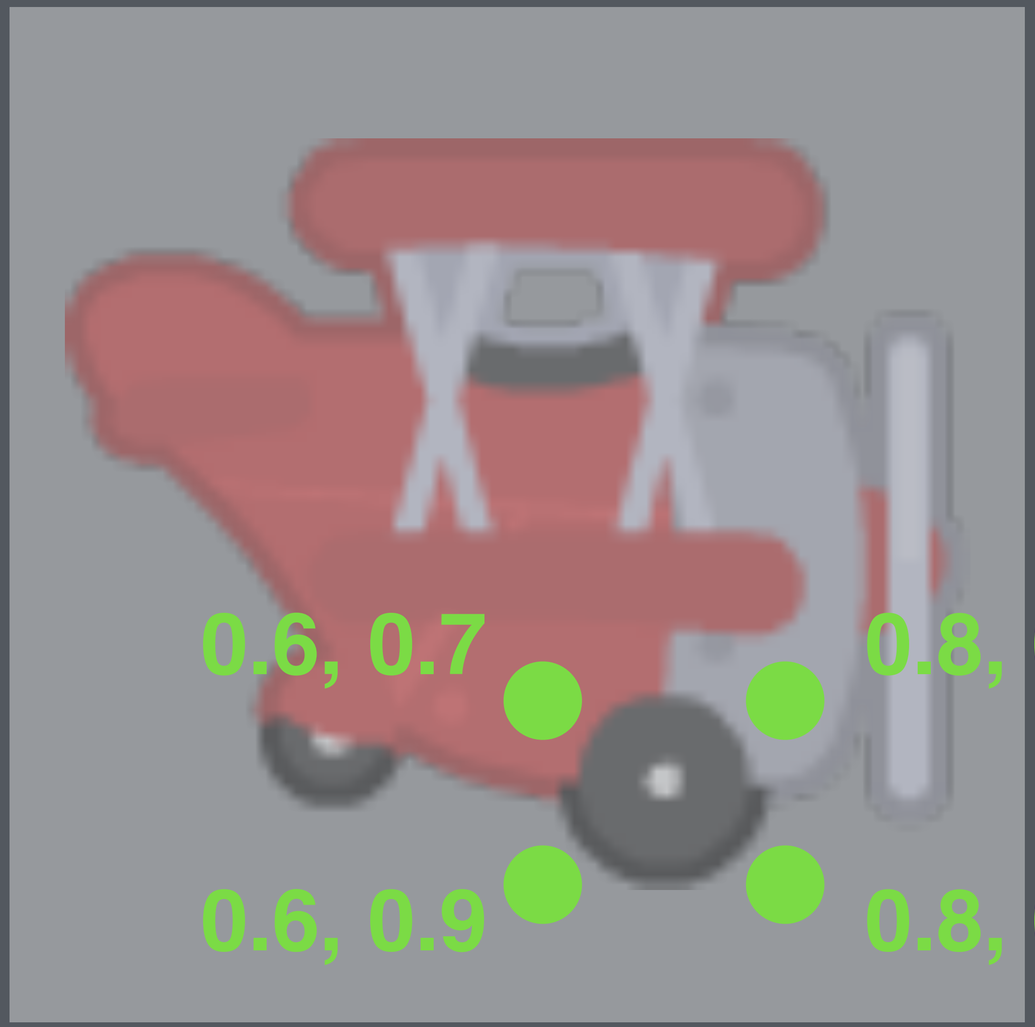


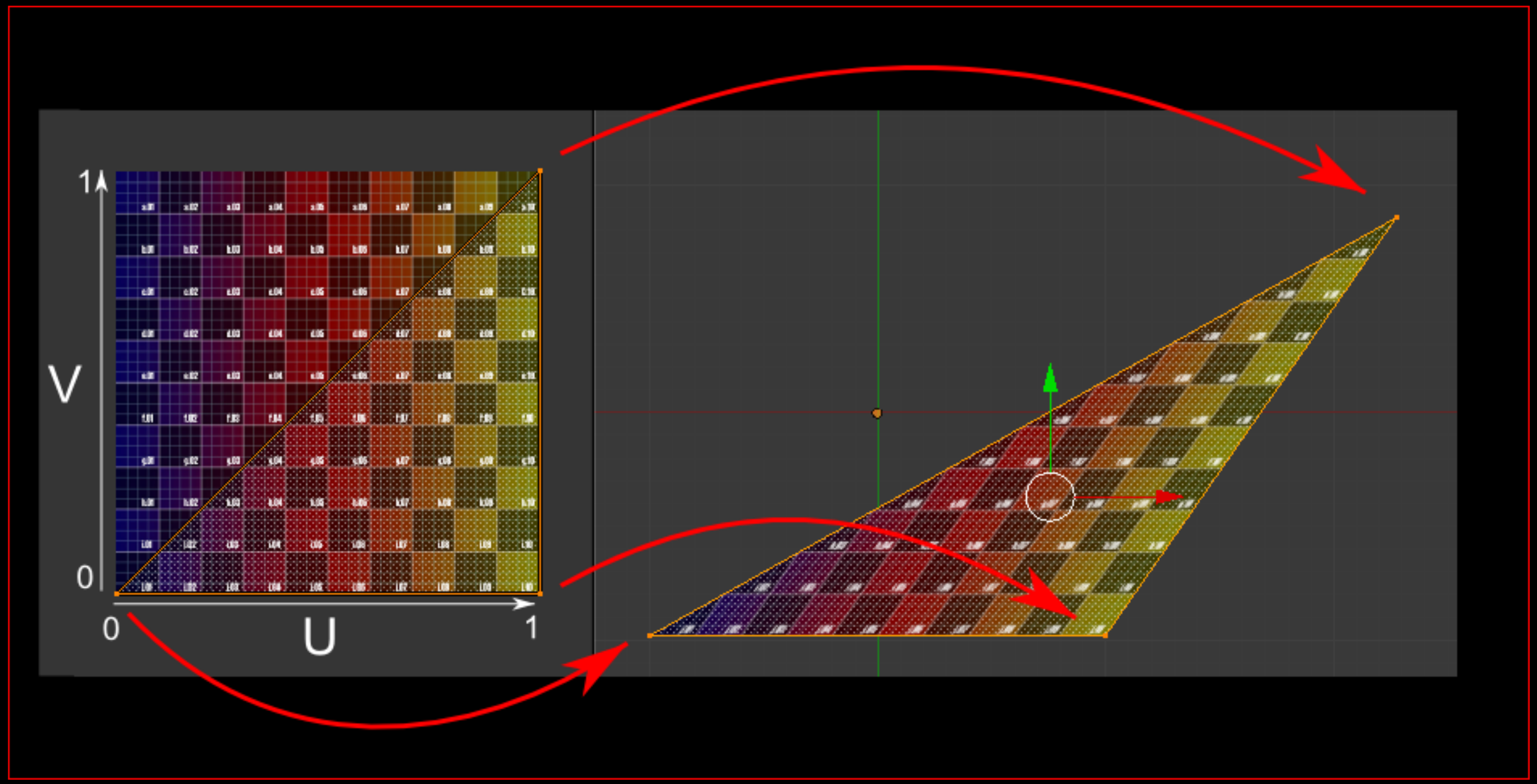
Texture coordinates are defined in 0-1 units called UV coordinates, not pixels!

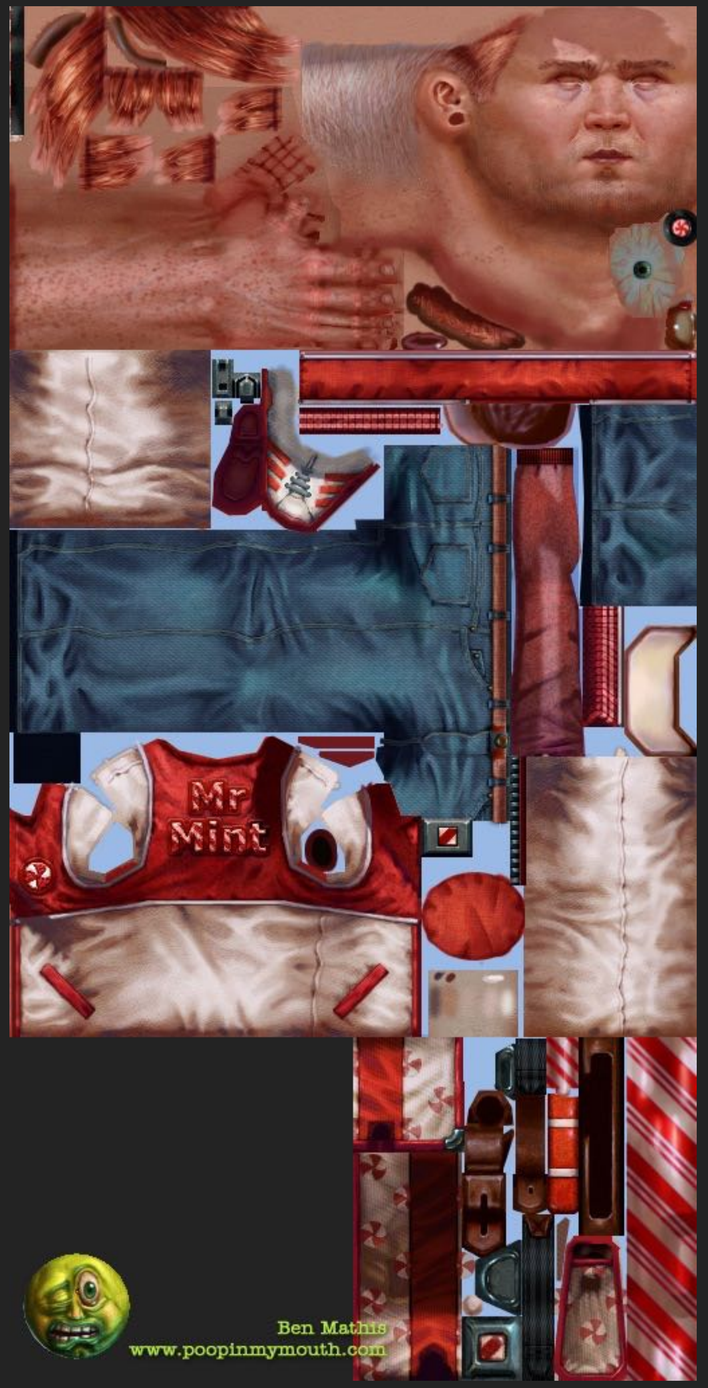
```
void glVertexAttribPointer (GLint index, GLint
size, GLenum type, GLboolean normalized, GLsizei
stride, const GLvoid *pointer);
```

Defines an array of **vertex data**.

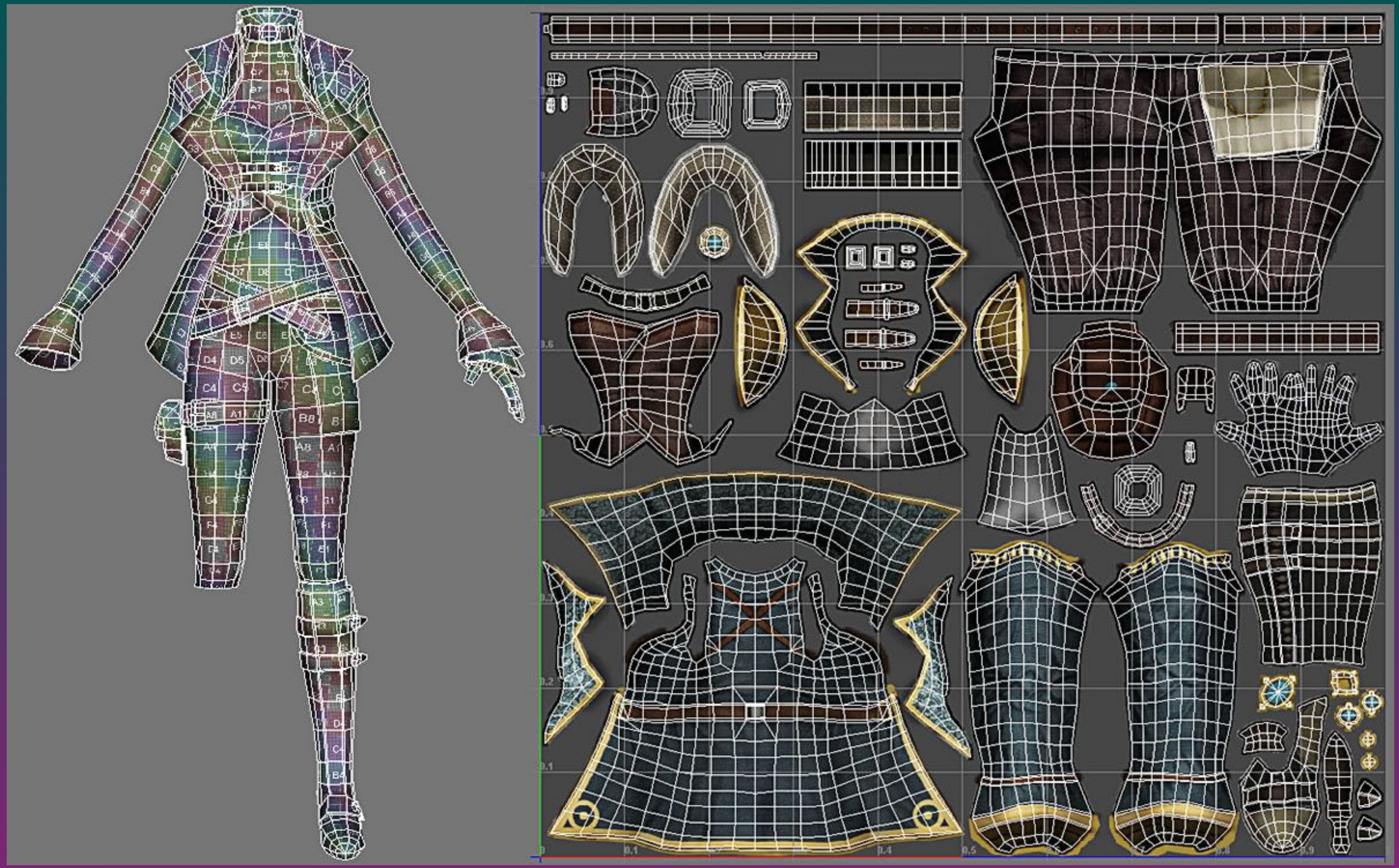
```
float texCoords[] = {0.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f};

glVertexAttribPointer(program.texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords);
```


0.6, 0.7
0.8, 0.7
0.6, 0.9
0.8, 0.9


1
V
0
0
U
1


Mr
Mint
Ben Mathis
www.poopinmymouth.com

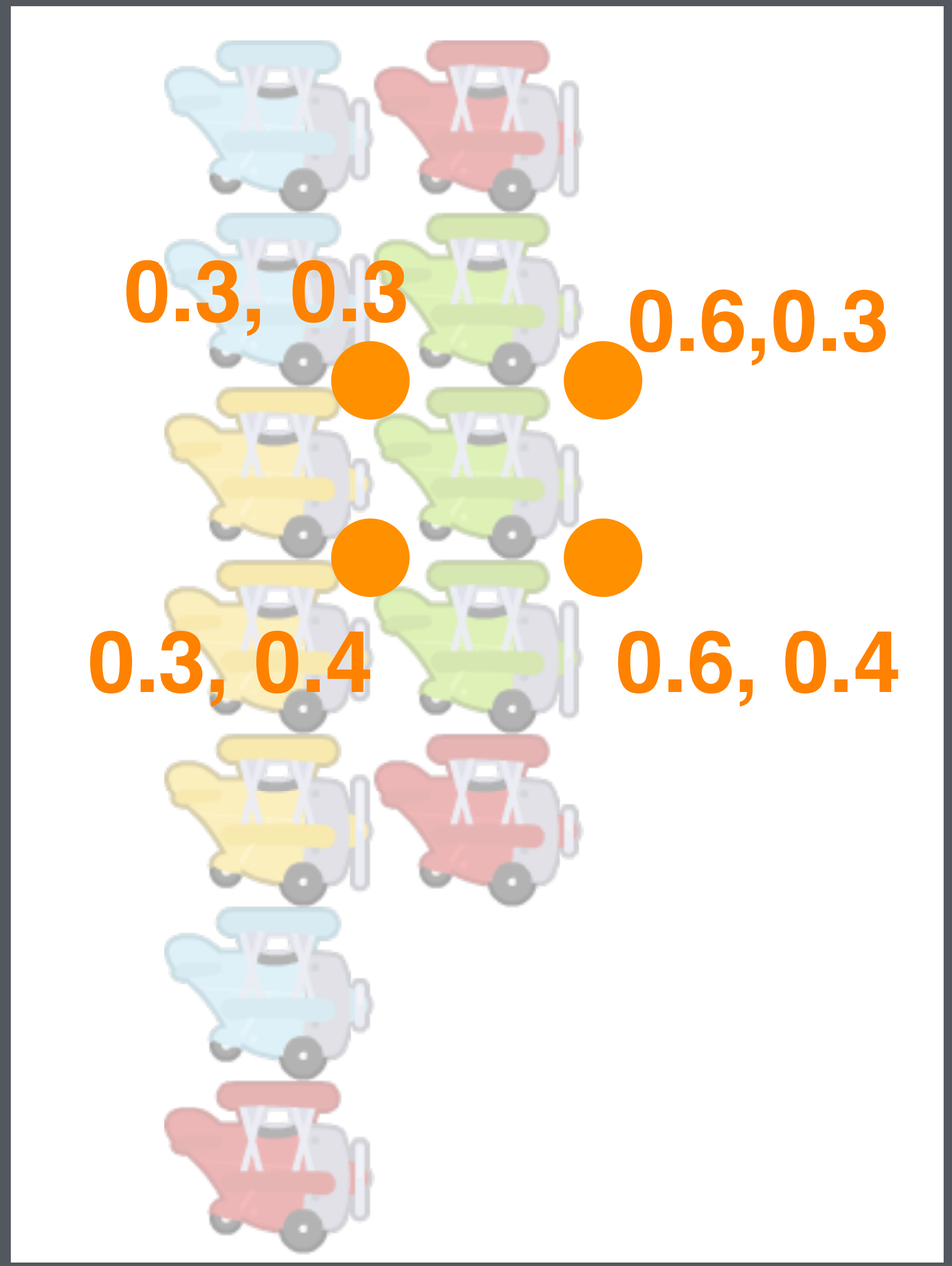

Ben Mathis
www.poopinmymouth.com

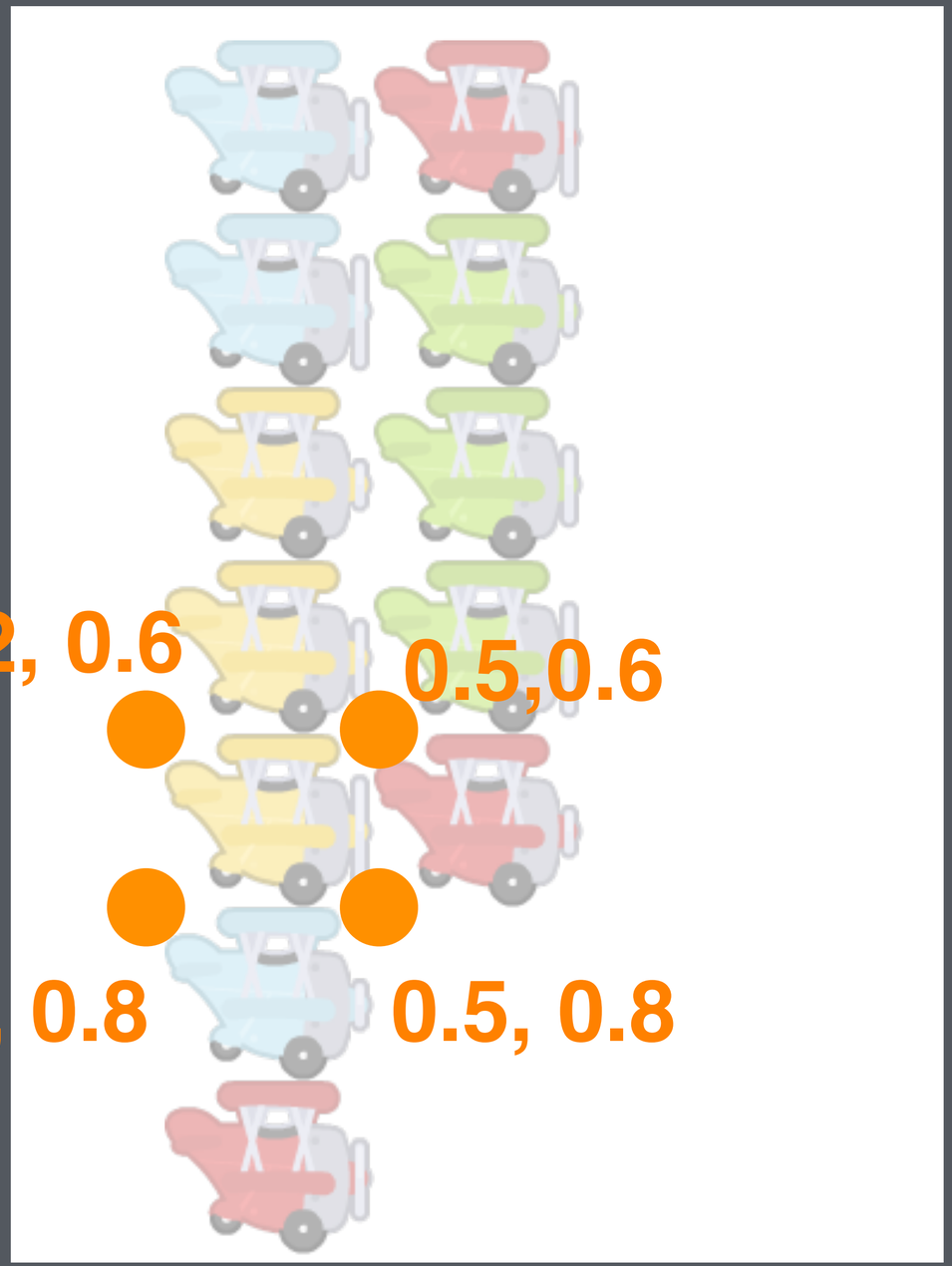


Using texture coordinates in 2D graphics.

Texture atlases.

A single texture that contains multiple sprites arranged in a single image.


0.3, 0.3
0.6,0.3
0.3, 0.4
0.6, 0.4


0.2, 0.6
0.5,0.6
0.2, 0.8
0.5, 0.8


TNT

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~ □
€ □ ‚ ƒ „ … † ‡ ˆ ‰ Š ‹ Œ □ Ž □
□ ‘ ’ “ ” • – — ˜ ™ š › œ □ ž Ÿ
  ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯
° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï
ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

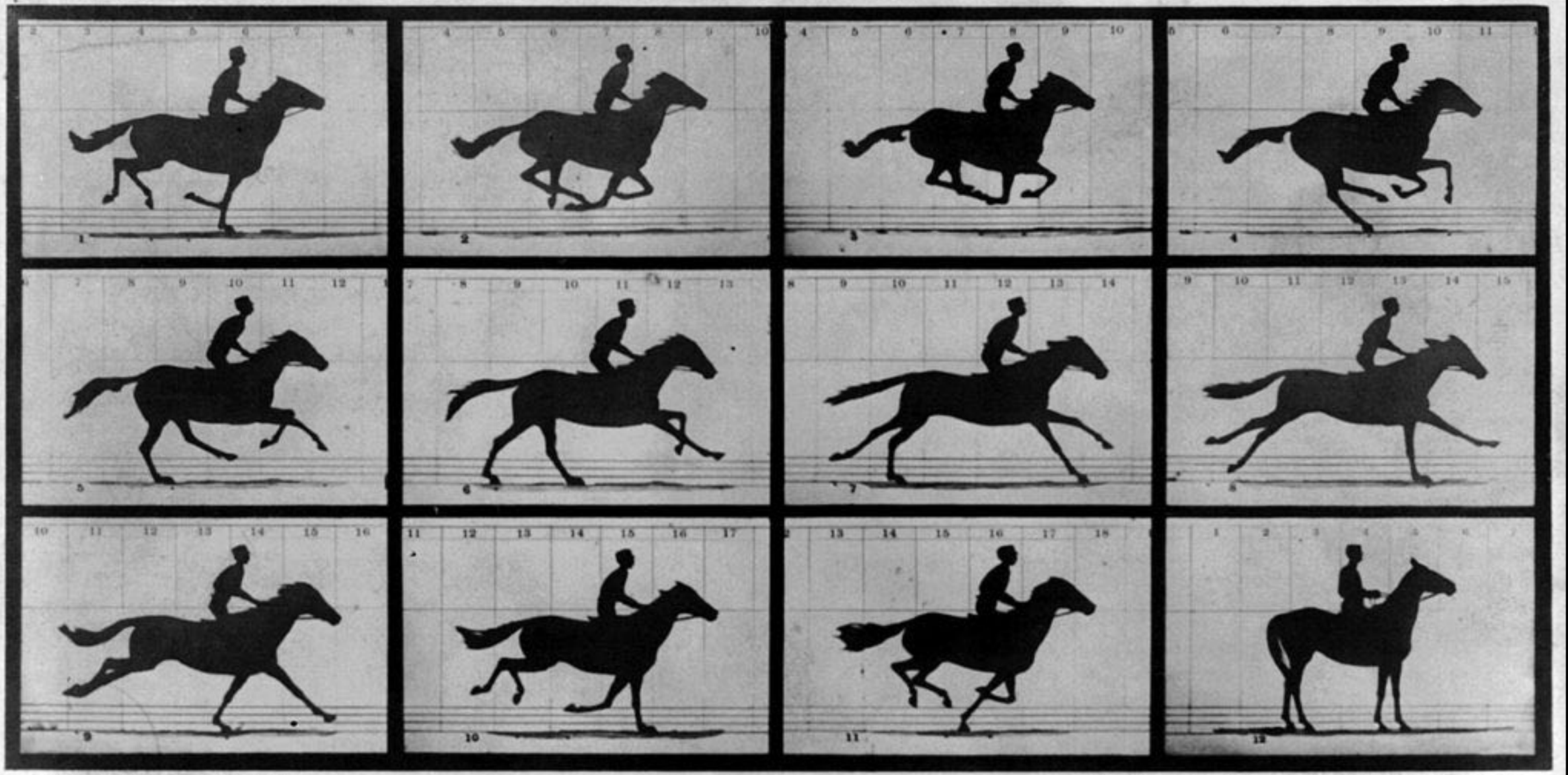
Evenly spaced sprite sheets


0
1
2
3
4
5
6
7
8
9
10
11
12
...


```
int index = 10;
int spriteCountX = 8;
int spriteCountY = 4;
float u = (float)(((int)index) % spriteCountX) / (float) spriteCountX;
float v = (float)(((int)index) / spriteCountX) / (float) spriteCountY;
float spriteWidth = 1.0/(float)spriteCountX;
float spriteHeight = 1.0/(float)spriteCountY;

GLfloat spriteUVs[] = { u, v,
                        u, v+spriteHeight,
                        u+spriteWidth, v+spriteHeight,
                        u+spriteWidth, v
};
```

```
void DrawSpriteSheetSprite(ShaderProgram &program, int index, int spriteCountX,
int spriteCountY) {

    float u = (float)(((int)index) % spriteCountX) / (float) spriteCountX;
    float v = (float)(((int)index) / spriteCountX) / (float) spriteCountY;
    float spriteWidth = 1.0/(float)spriteCountX;
    float spriteHeight = 1.0/(float)spriteCountY;

    float texCoords[] = {
        u, v+spriteHeight,
        u+spriteWidth, v,
        u, v,
        u+spriteWidth, v,
        u, v+spriteHeight,
        u+spriteWidth, v+spriteHeight
    };

   float vertices[] = {-0.5f, -0.5f, 0.5f, 0.5f, -0.5f, 0.5f, 0.5f, 0.5f,  -0.5f,
  -0.5f, 0.5f, -0.5f};

    // draw this data

}
```

Sprite animation



MORSE'S Gallery, 417 Montgomery St., San Francisco.

THE HORSE IN MOTION.

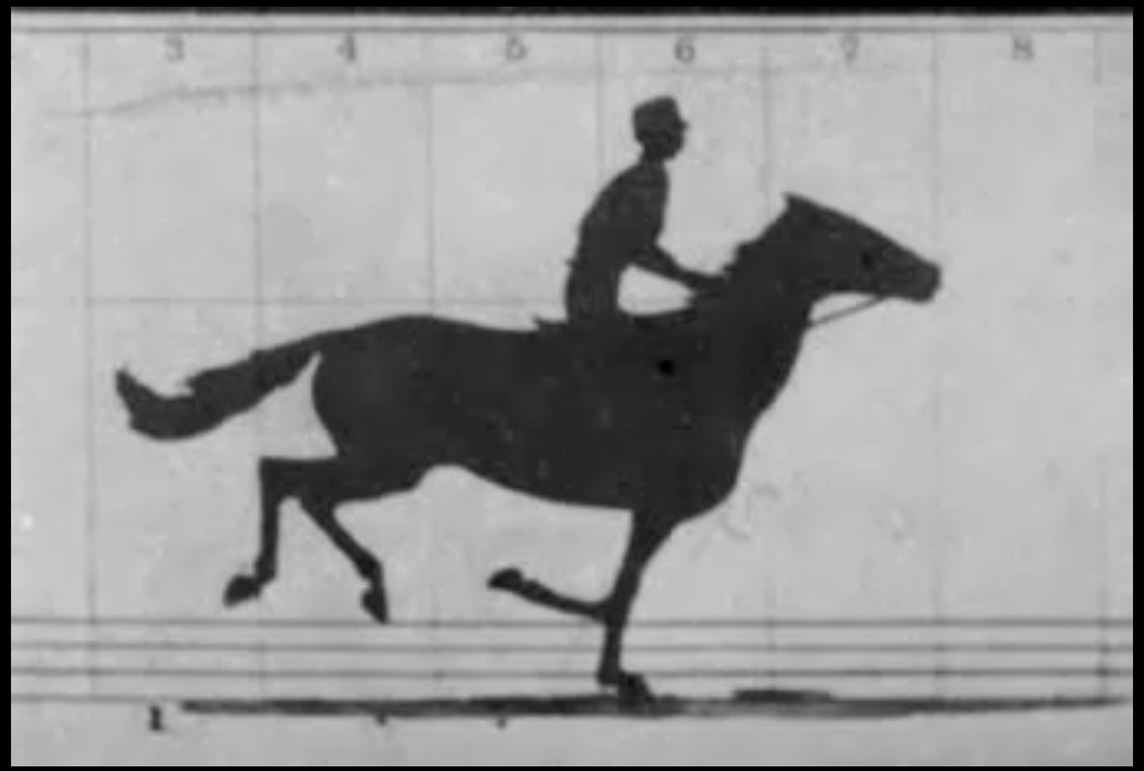
Illustrated by

MUYBRIDGE.

AUTOMATIC ELECTRO-PHOTOGRAPH.

"SALLIE GARDNER," owned by LELAND STANFORD; running at a 1.40 gait over the Palo Alto track, 19th June, 1878.

The negatives of these photographs were made at intervals of twenty-seven inches of distance, and about the twenty-fifth part of a second of time; they illustrate consecutive positions assumed in each twenty-seven inches of progress during a single stride of the mare. The vertical lines were twenty-seven inches apart; the horizontal lines represent elevations of four inches each. The exposure of each negative was less than the two-thousandth part of a second.


3
4
5
6
7
8
1

1. Define indices of an animation (e.g. 0-6)
2. Keep a timer
3. Go to next frame when timer hits desired value.
4. If at the last frame, go to first frame (if looped animation).

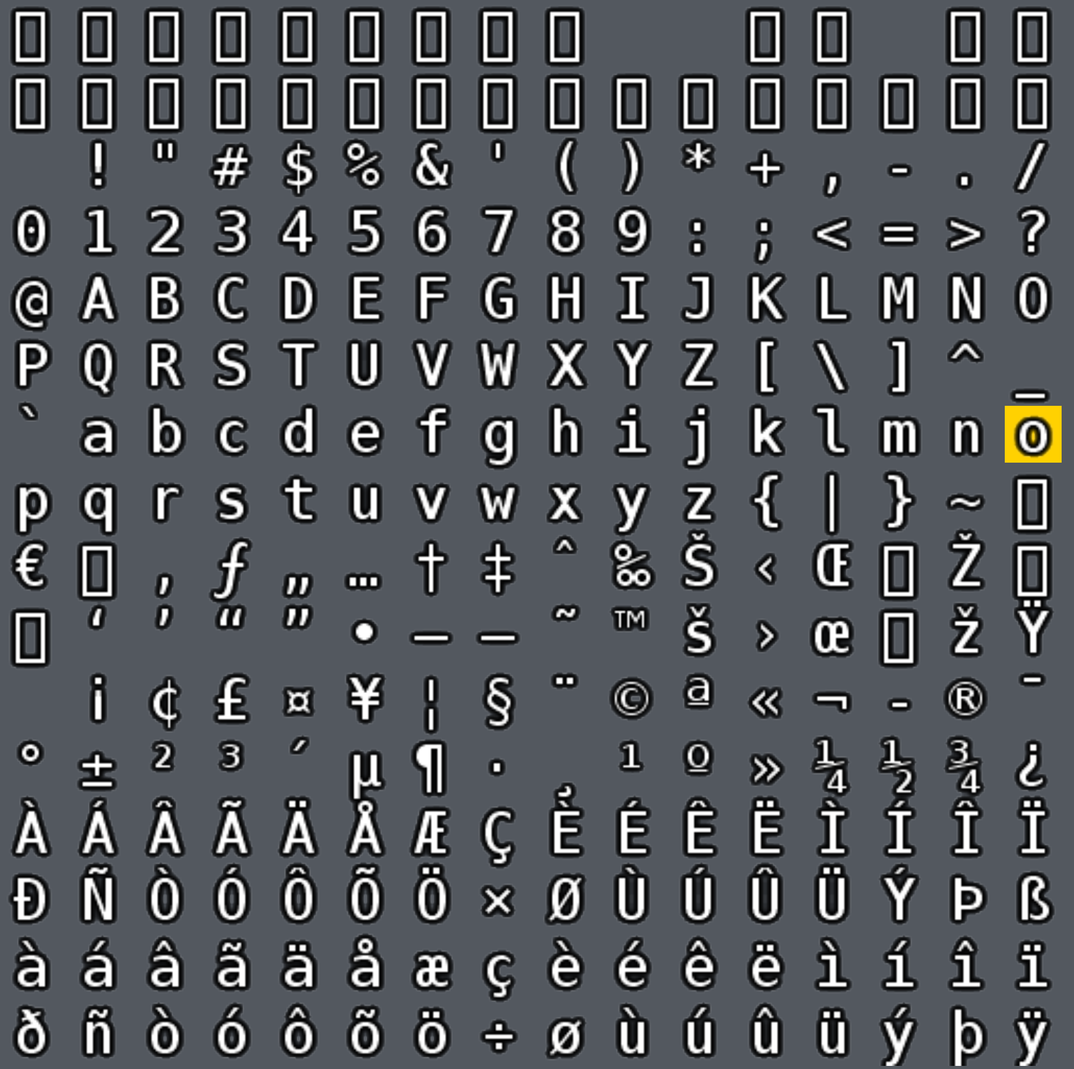
```
const int runAnimation[] = {9, 10, 11, 12, 13};
const int numFrames = 5;
float animationElapsed = 0.0f;
float framesPerSecond = 30.0f;
int currentIndex = 0;
```

In our **loop**:

```
animationElapsed += elapsed;

if(animationElapsed > 1.0/framesPerSecond) {
    currentIndex++;
    animationElapsed = 0.0;

    if(currentIndex > numFrames-1) {
                currentIndex = 0;
    }
}

DrawSpriteSheetSprite(runAnimation[currentIndex], 8, 4);
```

Monospaced font rendering.

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~ □
€ □ ‚ ƒ „ … † ‡ ˆ ‰ Š ‹ Œ □ Ž □
□ ‘ ’ “ ” • – — ˜ ™ š › œ □ ž Ÿ
  ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯
° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï
ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ



To render a string, we must look at it character by character and draw 2 triangles for each letter using the appropriate UV coordinates.

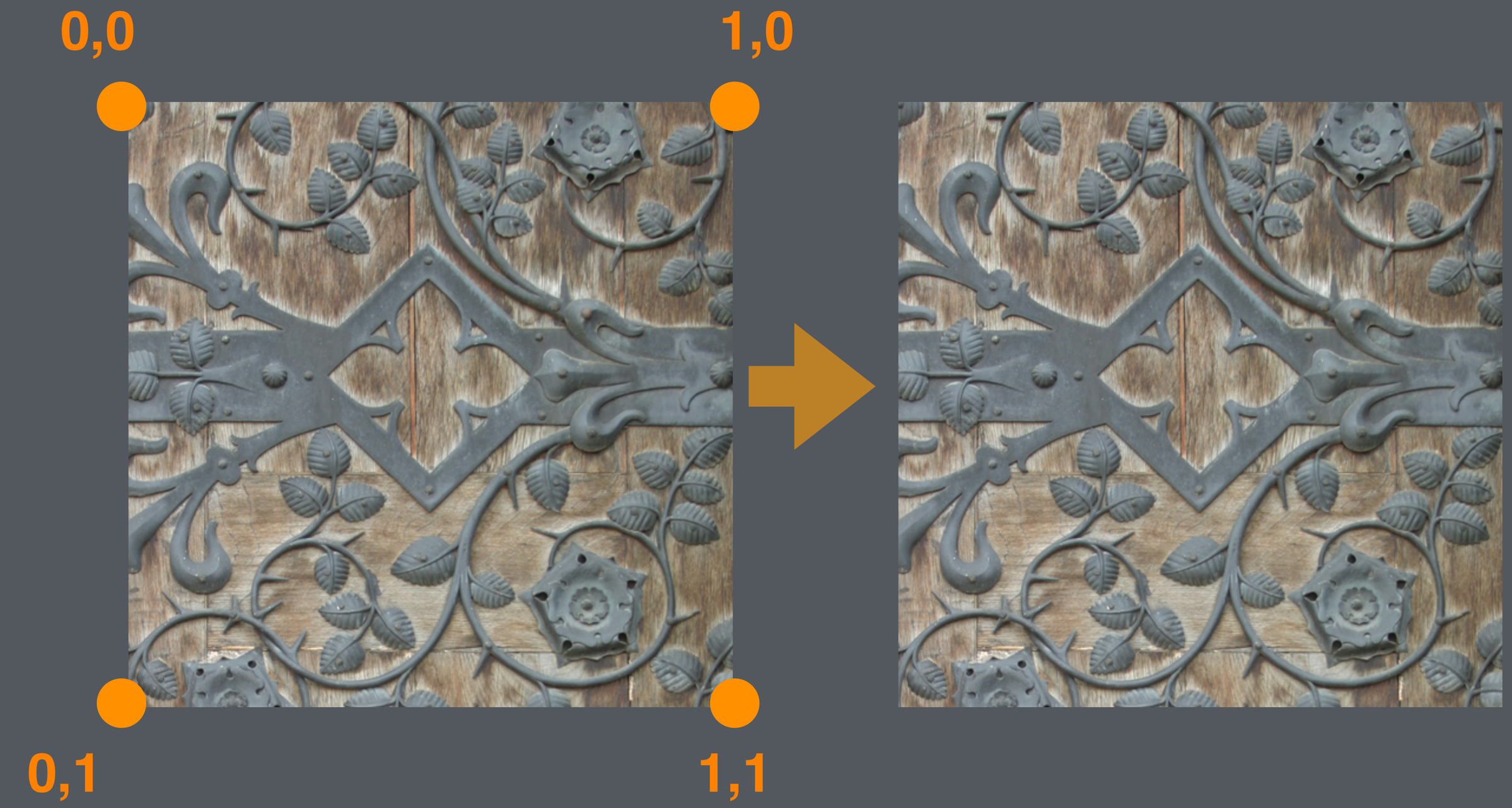
```
string test = "This is a string!";
cout << (int)test[3] << endl; // prints 115, which is 's'
```

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

```
void DrawText(ShaderProgram *program, int fontTexture, std::string text, float size, float spacing) {

    float texture_size = 1.0/16.0f;

    std::vector<float> vertexData;
    std::vector<float> texCoordData;

    for(int i=0; i < text.size(); i++) {

         int spriteIndex = (int)text[i];

        float texture_x = (float)(spriteIndex % 16) / 16.0f;
        float texture_y = (float)(spriteIndex / 16) / 16.0f;

        vertexData.insert(vertexData.end(), {
            ((size+spacing) * i) + (-0.5f * size), 0.5f * size,
            ((size+spacing) * i) + (-0.5f * size), -0.5f * size,
            ((size+spacing) * i) + (0.5f * size), 0.5f * size,
            ((size+spacing) * i) + (0.5f * size), -0.5f * size,
            ((size+spacing) * i) + (0.5f * size), 0.5f * size,
            ((size+spacing) * i) + (-0.5f * size), -0.5f * size,
        });
        texCoordData.insert(texCoordData.end(), {
            texture_x, texture_y,
            texture_x, texture_y + texture_size,
            texture_x + texture_size, texture_y,
            texture_x + texture_size, texture_y + texture_size,
            texture_x + texture_size, texture_y,
            texture_x, texture_y + texture_size,
        });
    }

    glBindTexture(GL_TEXTURE_2D, fontTexture);

    // draw this data (use the .data() method of std::vector to get pointer to data)
    // draw this yourself, use text.size() * 6 or vertexData.size()/2 to get number of vertices
}
```

Texture wrap modes.


0,0
1,0
0,1
1,1

0,0

2,0

0,2

2,2

Repeat

0,2

2,2

0,0

2,0

Clamp

0,2

2,2

0,0

2,0

```
void glTexParameteri (GLenum target, GLenum pname,
GLint param);
```

Sets a texture parameter of the specified texture target.

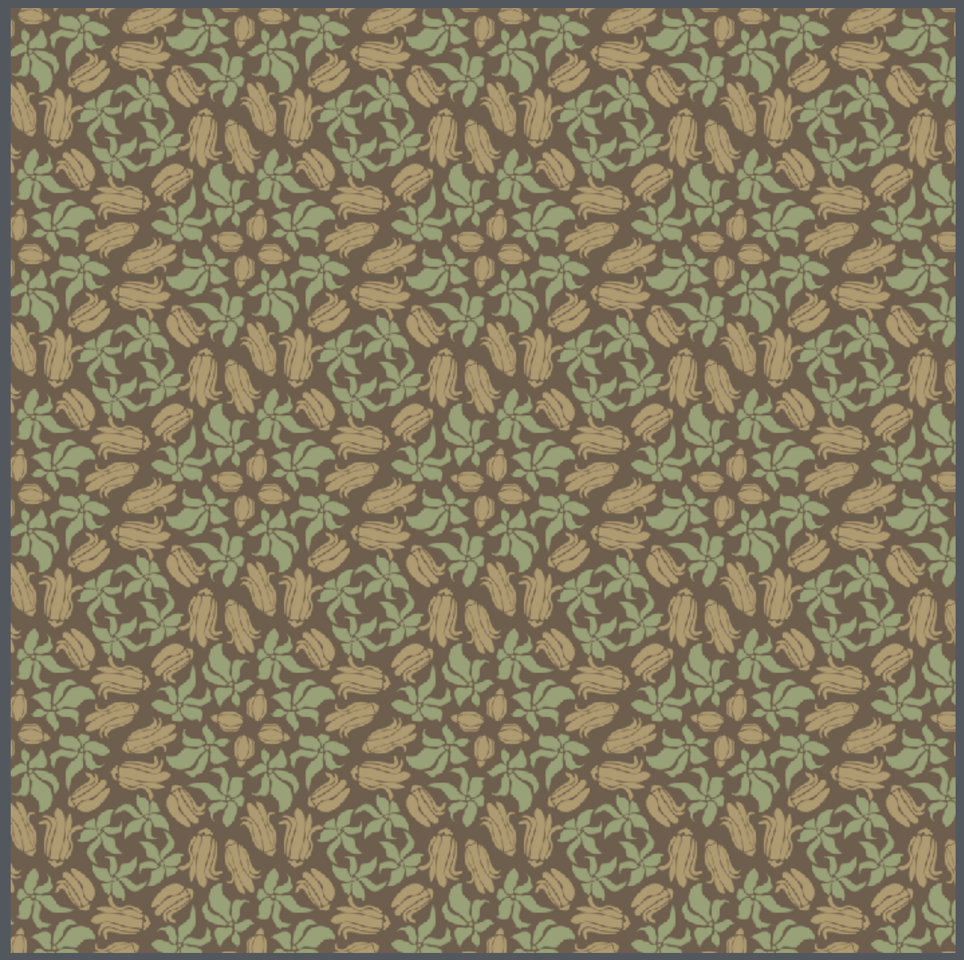
CLAMPING

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

REPEATING

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

Use GL_REPEAT for tiling textures



Use GL_CLAMP for non-tiled images with alpha.