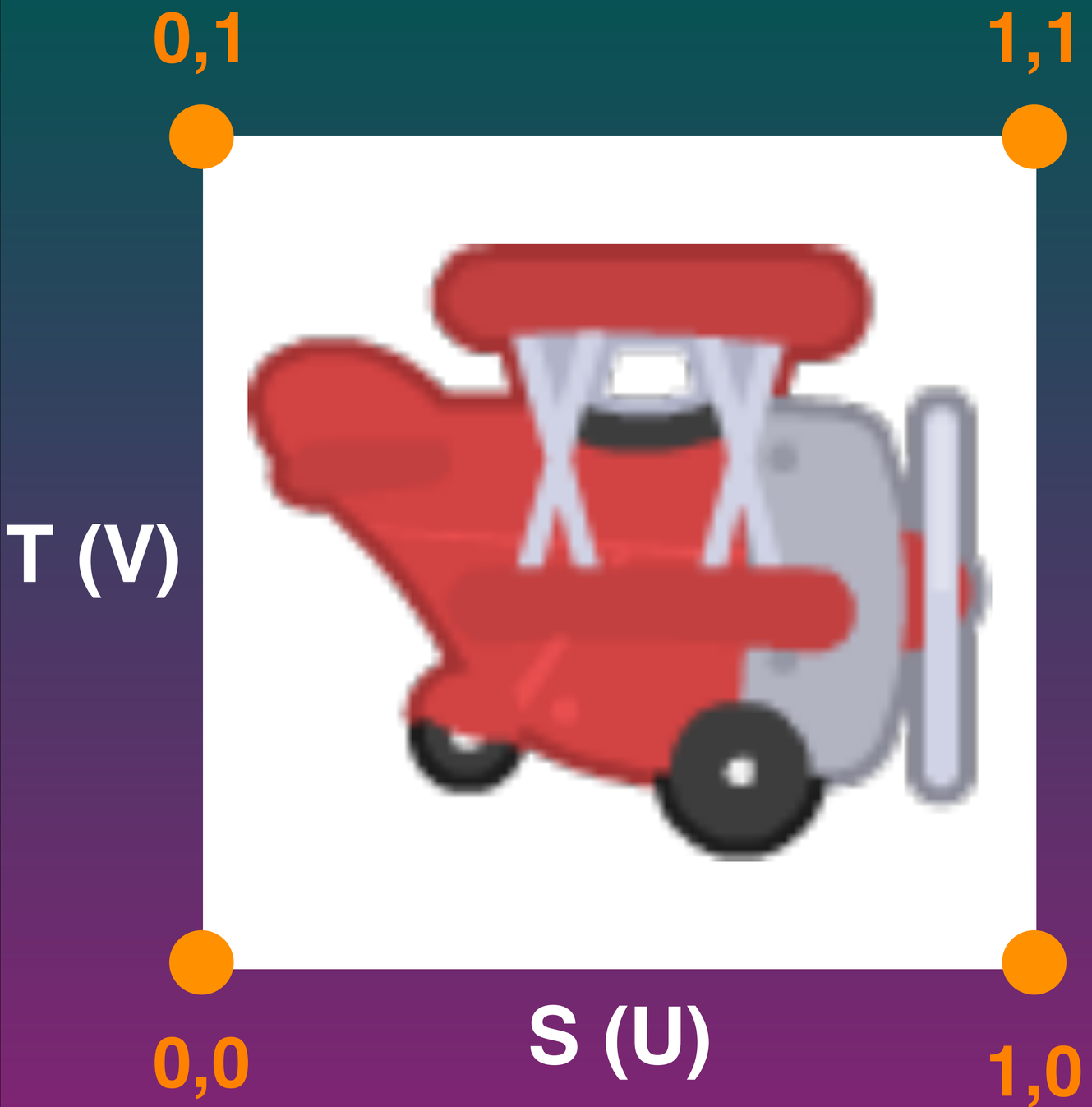
Graphics Foundations

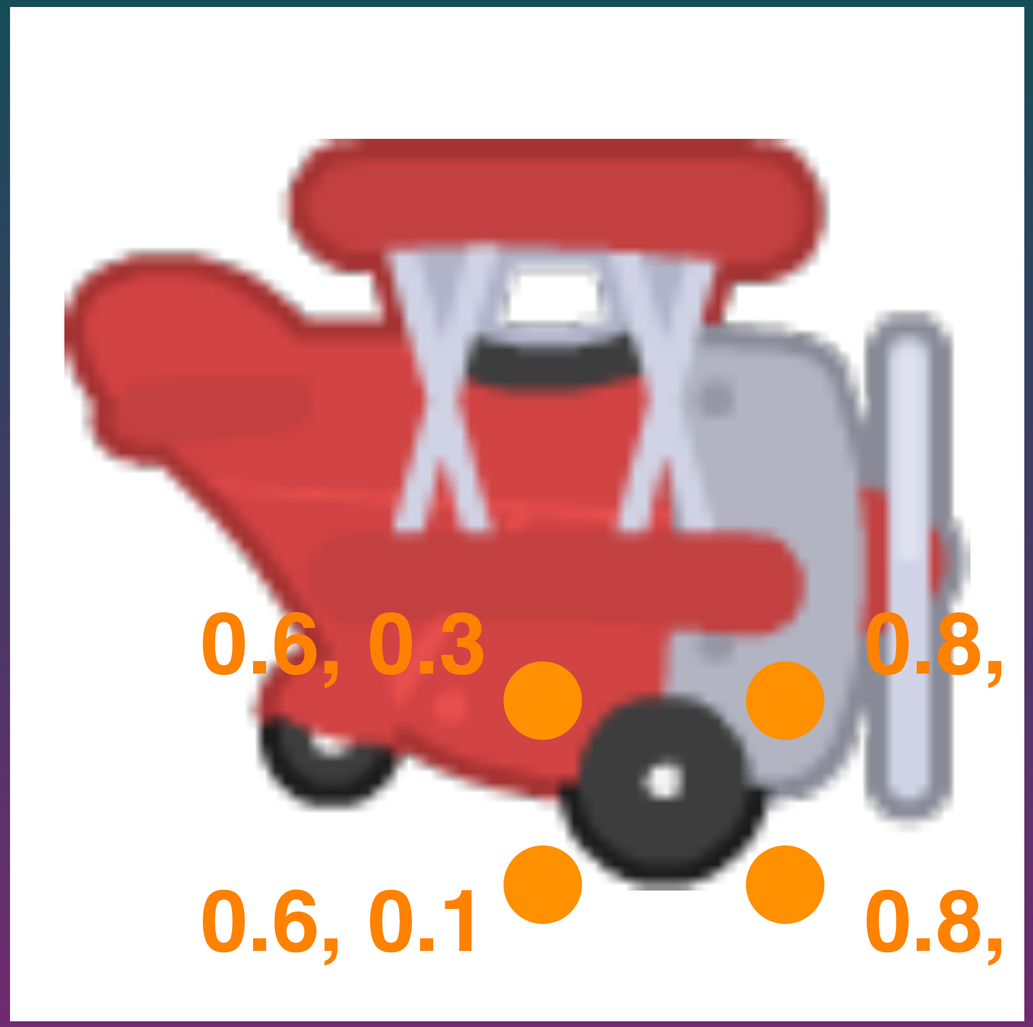
Part 3

Texture coordinates



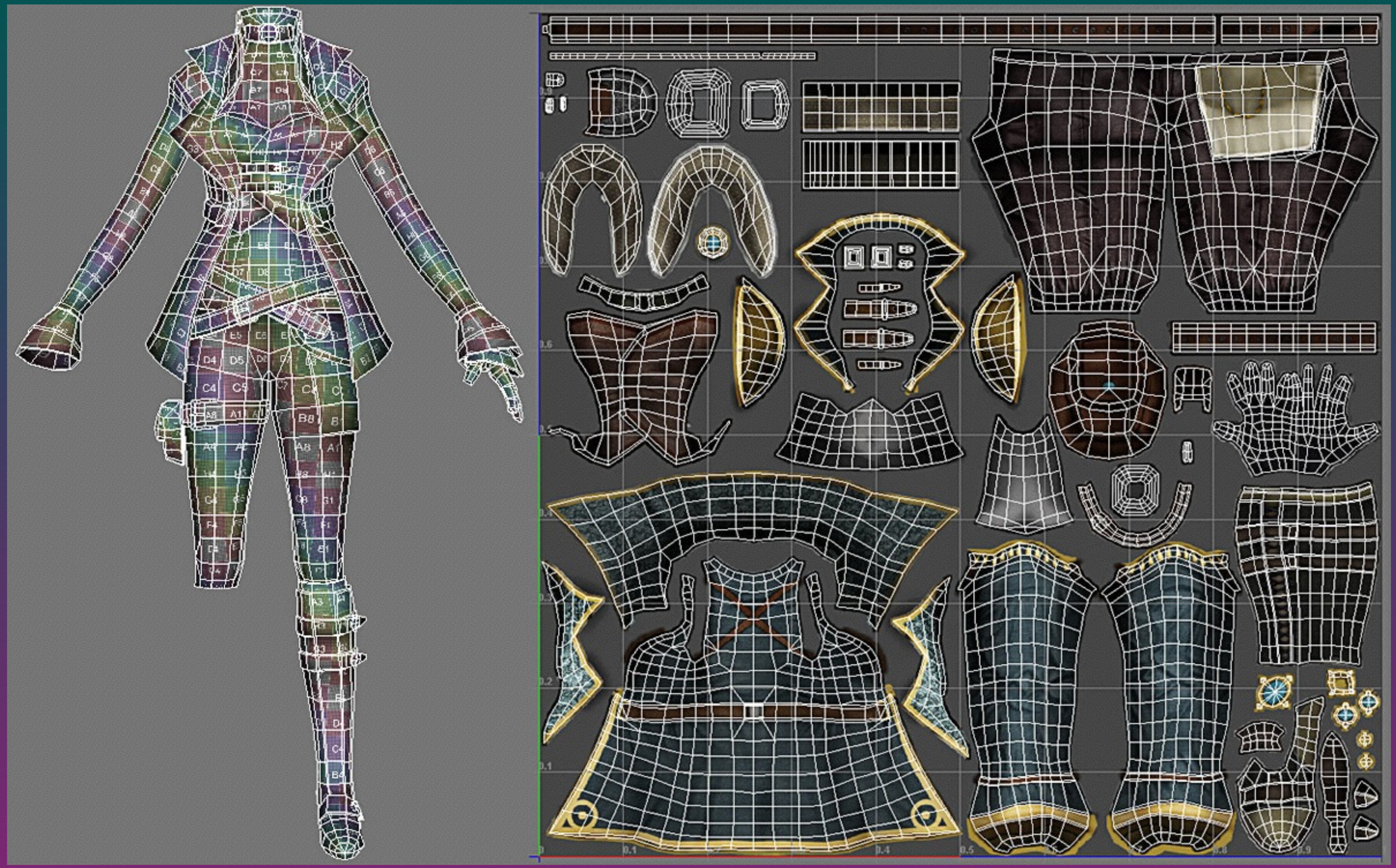
Texture coordinates are defined in 0-1 units called UV coordinates, not pixels!

```
GLfloat quadUVs[] = {0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0};
glTexCoordPointer(2, GL_FLOAT, 0, quadUVs);
```


0.6, 0.3
0.8, 0.3
0.6, 0.1
0.8, 0.1

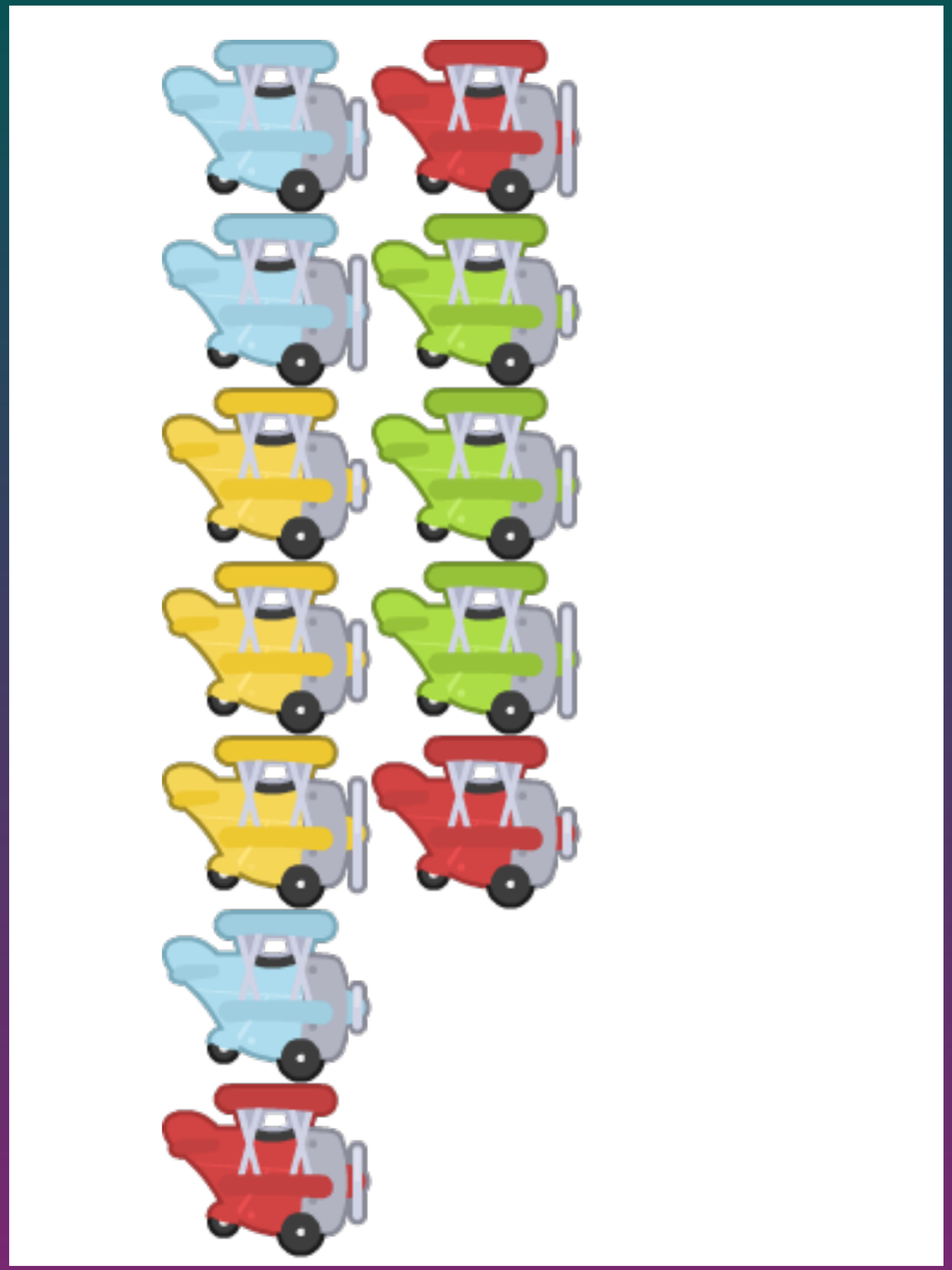

Mr
Mint
Ben Mathis
www.poopinmymouth.com


Ben Mathis
www.poopinmymouth.com

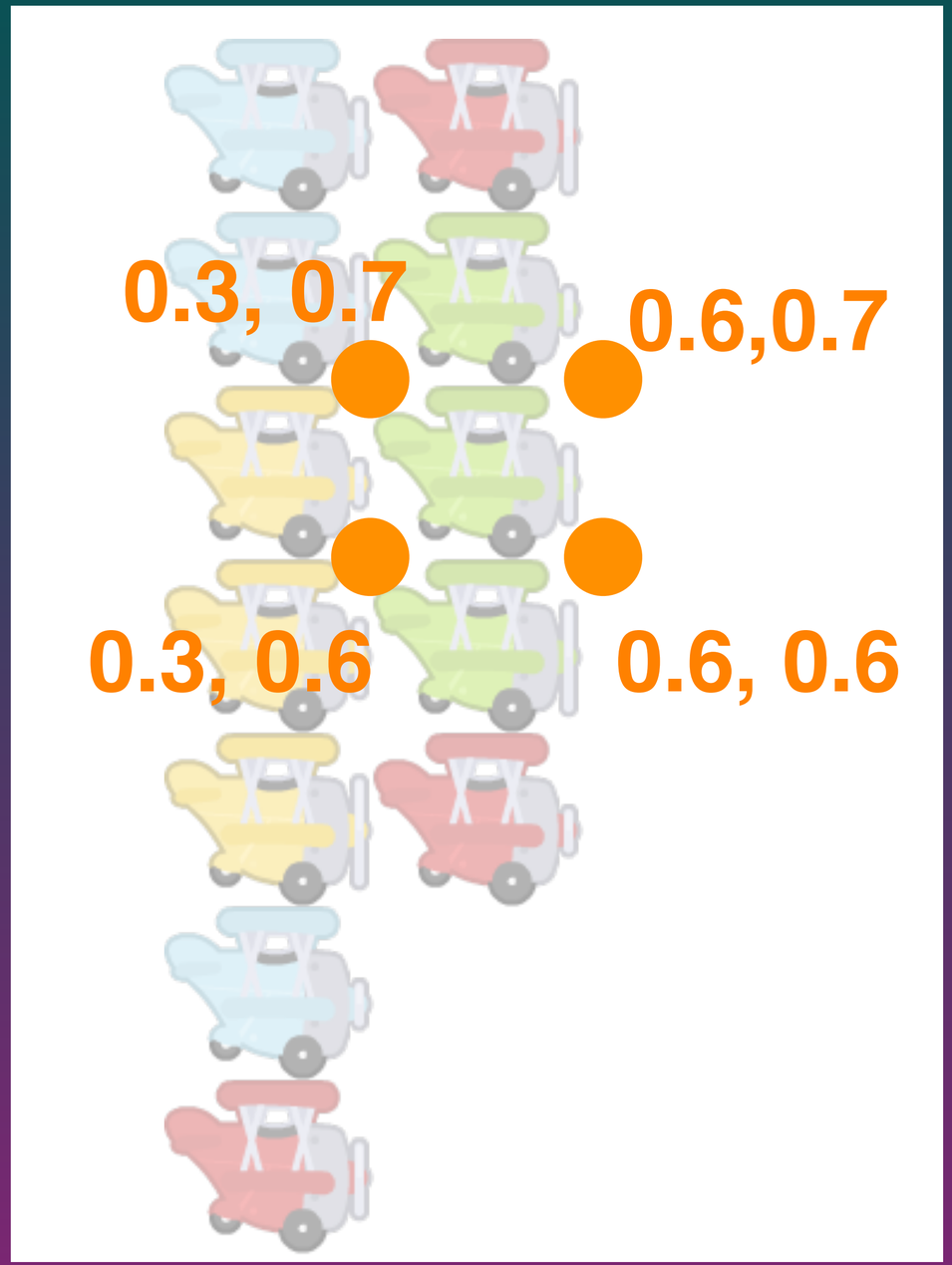


Texture coordinates in 2D graphics.

Texture atlas.



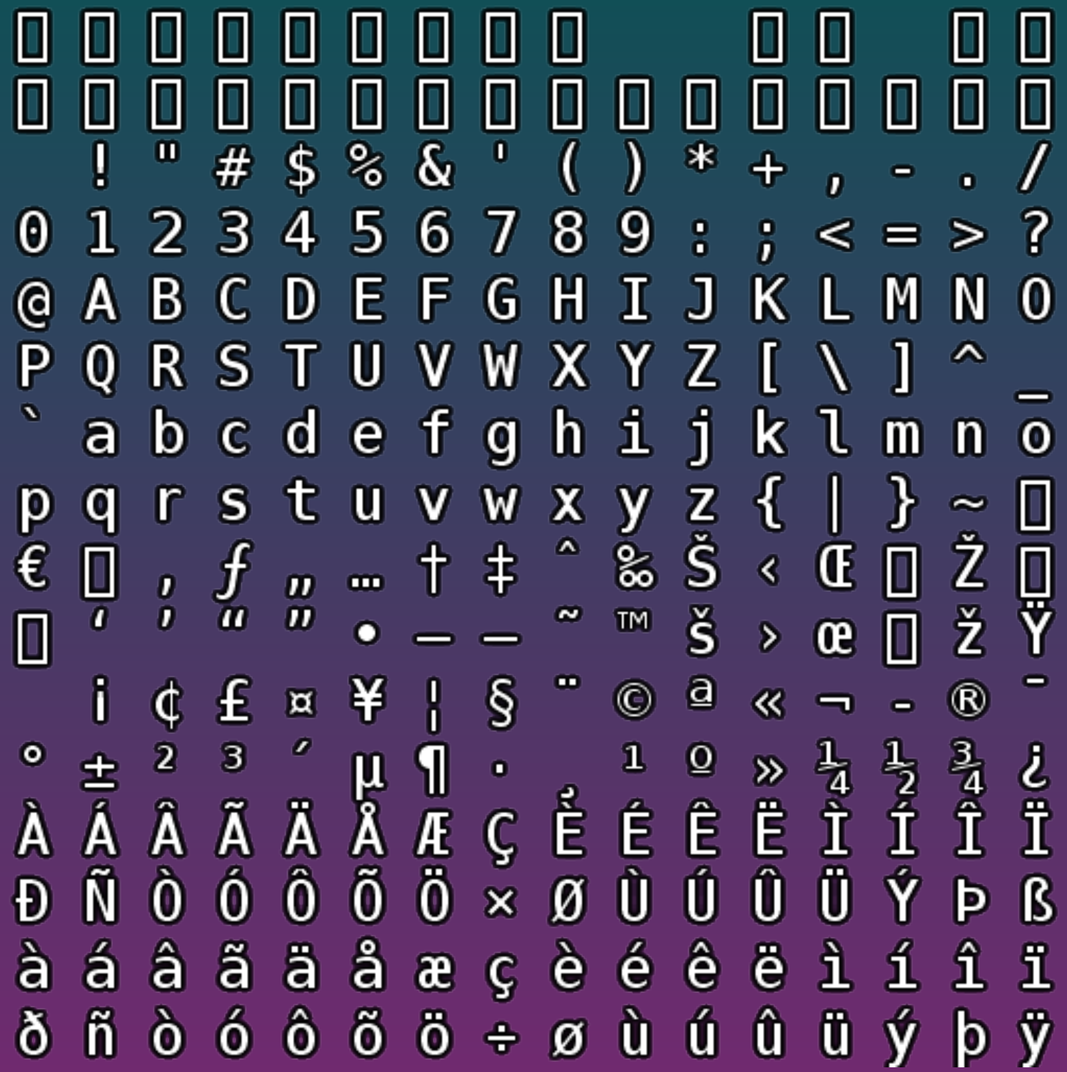
A single texture that contains multiple sprites arranged in a single image.


0.3, 0.7
0.6,0.7
0.3, 0.6
0.6, 0.6


0.2, 0.4
0.5,0.4
0.2, 0.3
0.5, 0.3


TNT


Back
4 x 5
alt: pick
ctrl: erase
shift: axis



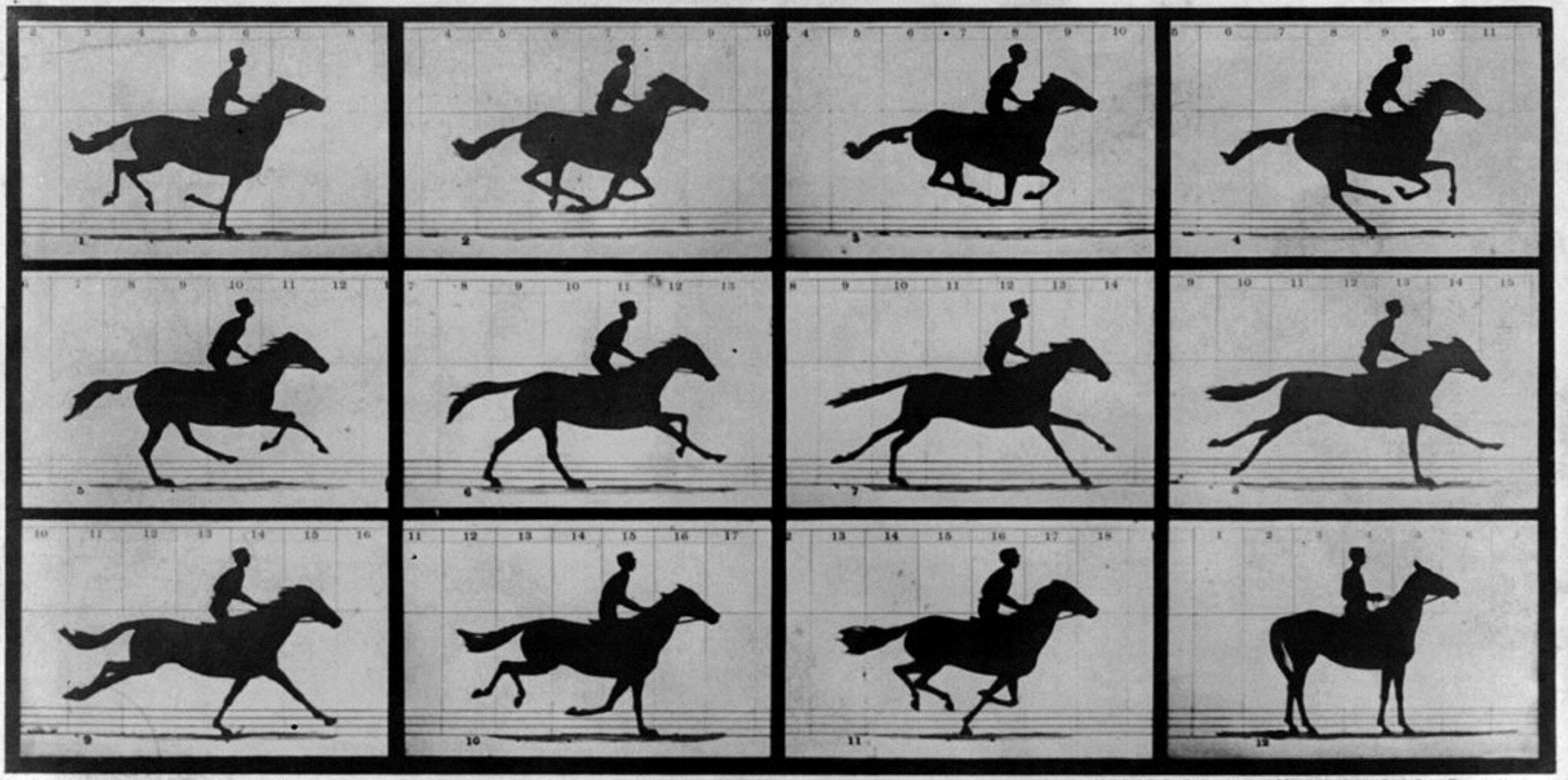
Evenly spaced sprite sheets


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

```
void DrawSpriteSheetSprite(int spriteTexture, int index, int spriteCountX, int
spriteCountY) {

   // our regular sprite drawing

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

    // our regular sprite drawing

}
```

Sprite animation



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

THE HORSE IN MOTION.

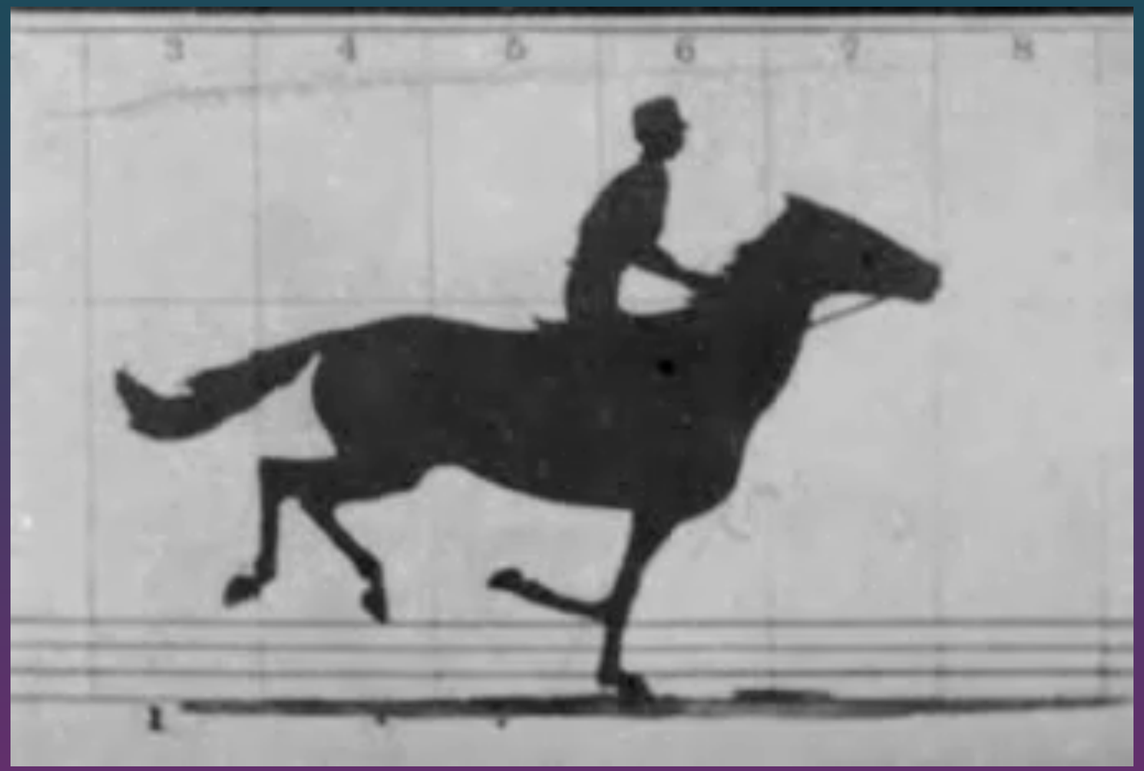
Illustrated by

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

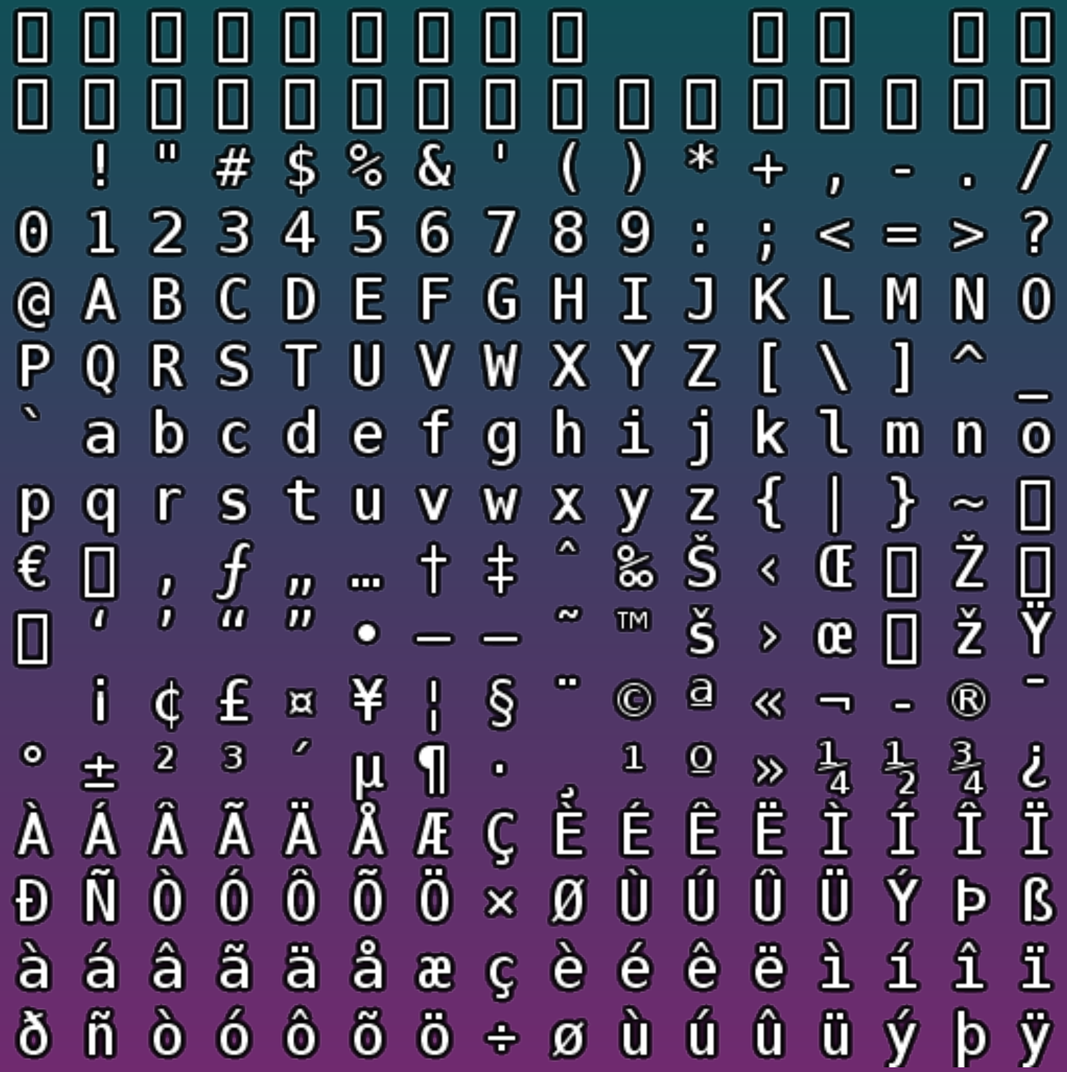
1. Define indices of an animation (e.g. 0-6)
2. Keep a timer
3. Go to next frame when timer hits desired value.
4. If at the last frame, go to first frame (if looped animation).

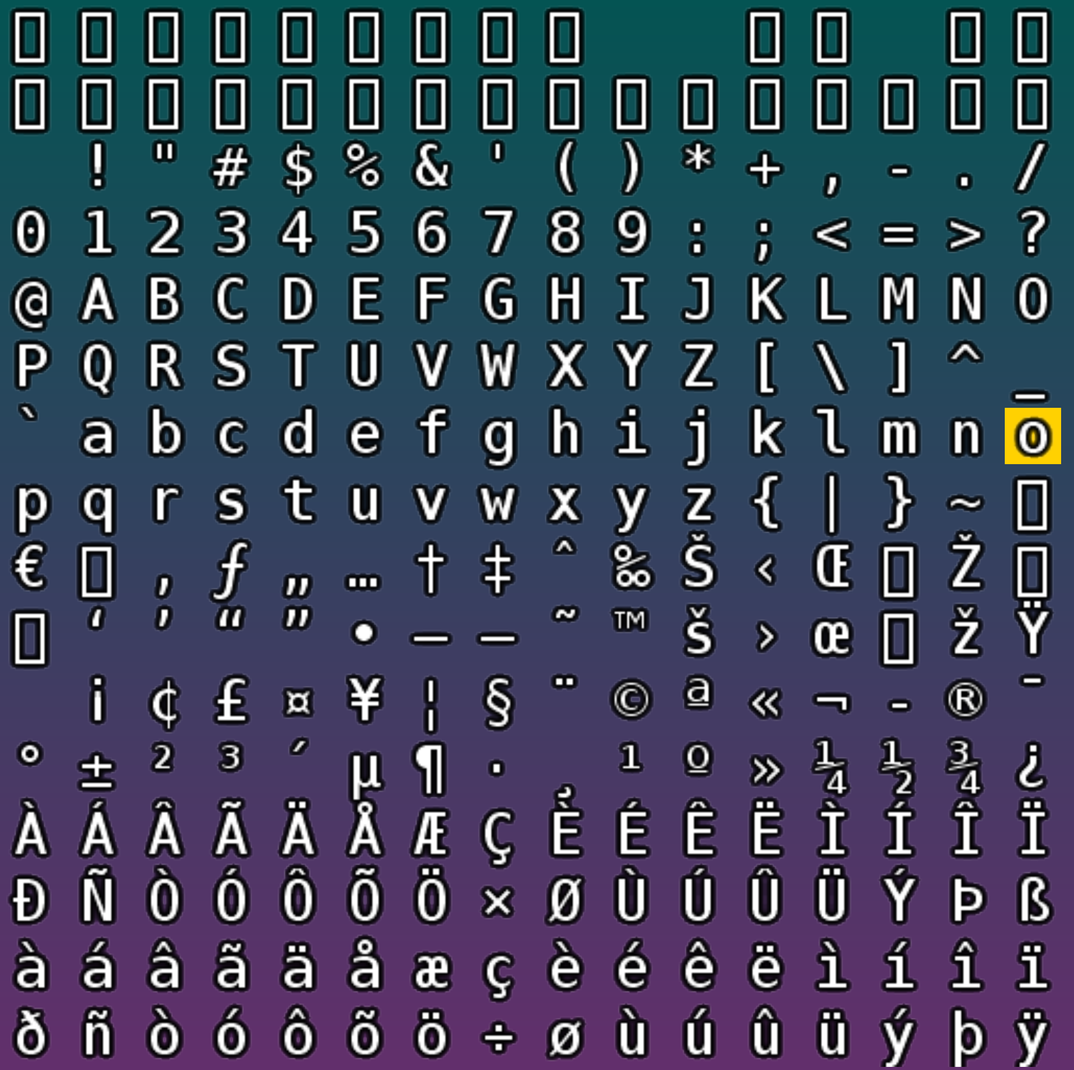
```
const int runAnimation[] = {9, 10, 11, 12, 13};
const int numFrames = 5;
float animationElapsed = 0.0f;
float framesPerSecond = 30.0f;
```

In our loop:

```
animationElapsed += elapsed;

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

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

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

Monospaced font rendering.





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

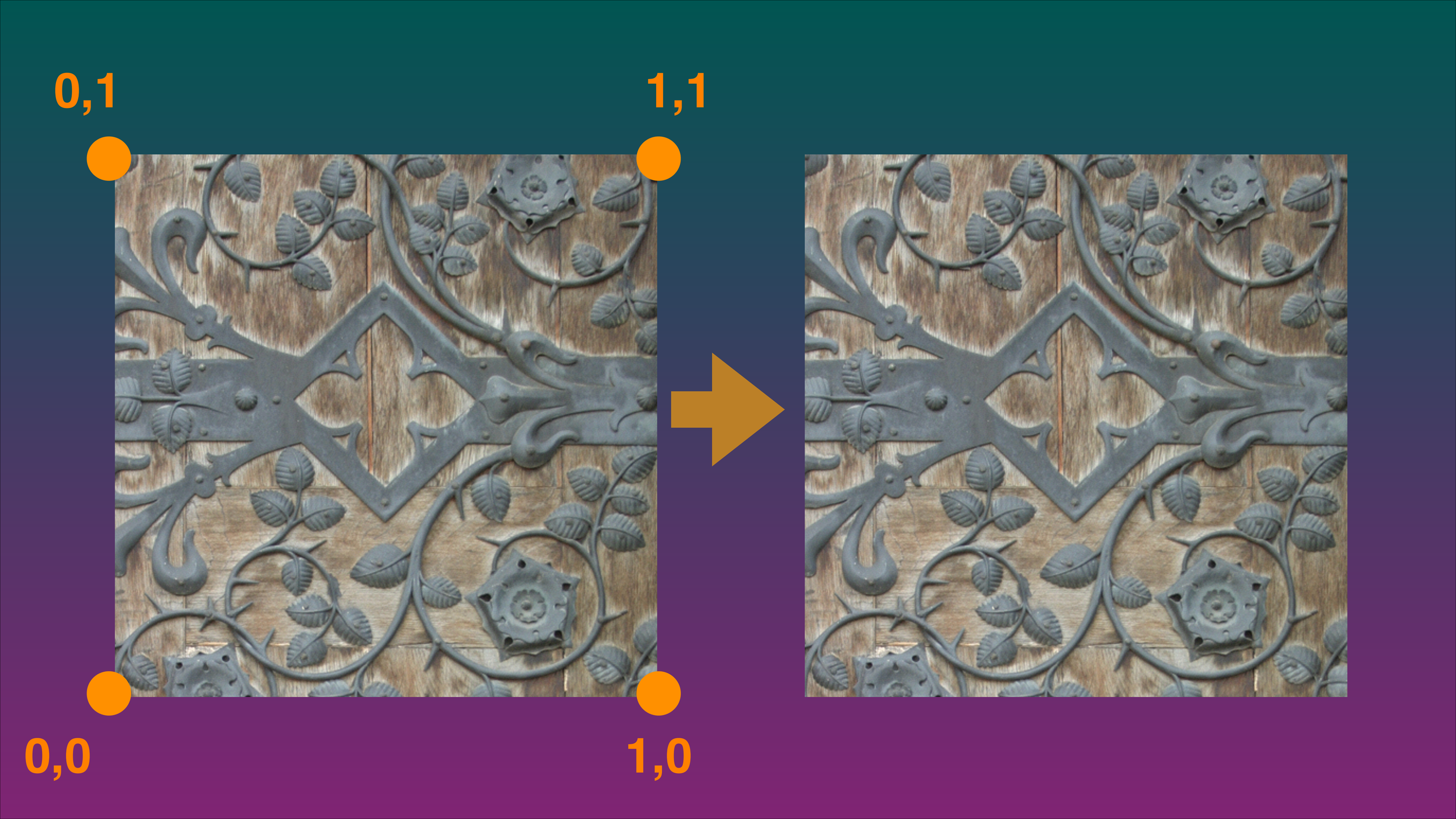
```
string test = "This is a string!";
cout << (int)test[3] << endl; // prints 115, which is 's'
```

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

Source: www.LookupTables.com

```
void DrawText(int fontTexture, string text, float size, float spacing, float r, float g, float b, float a) {

    glBindTexture(GL_TEXTURE_2D, fontTexture);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    float texture_size = 1.0/16.0f;

    vector<float> vertexData;
    vector<float> texCoordData;
    vector<float> colorData;

    for(int i=0; i < text.size(); i++) {
        float texture_x = (float)(((int)text[i]) % 16) / 16.0f;
        float texture_y = (float)(((int)text[i]) / 16) / 16.0f;

        colorData.insert(colorData.end(), {r,g,b,a, r,g,b,a, r,g,b,a, r,g,b,a});

        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});

        texCoordData.insert(texCoordData.end(), {texture_x, texture_y, texture_x, texture_y + texture_size, texture_x +
texture_size, texture_y + texture_size, texture_x + texture_size, texture_y});
    }

    glColorPointer(4, GL_FLOAT, 0, colorData.data());
    glEnableClientState(GL_COLOR_ARRAY);
    glVertexPointer(2, GL_FLOAT, 0, vertexData.data());
    glEnableClientState(GL_VERTEX_ARRAY);
    glTexCoordPointer(2, GL_FLOAT, 0, texCoordData.data());
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    glDrawArrays(GL_QUADS, 0, text.size() * 4);
}
```

Texture wrap modes.


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

0,2

2,2

?

0,0

2,0

Repeat

0,2

2,2

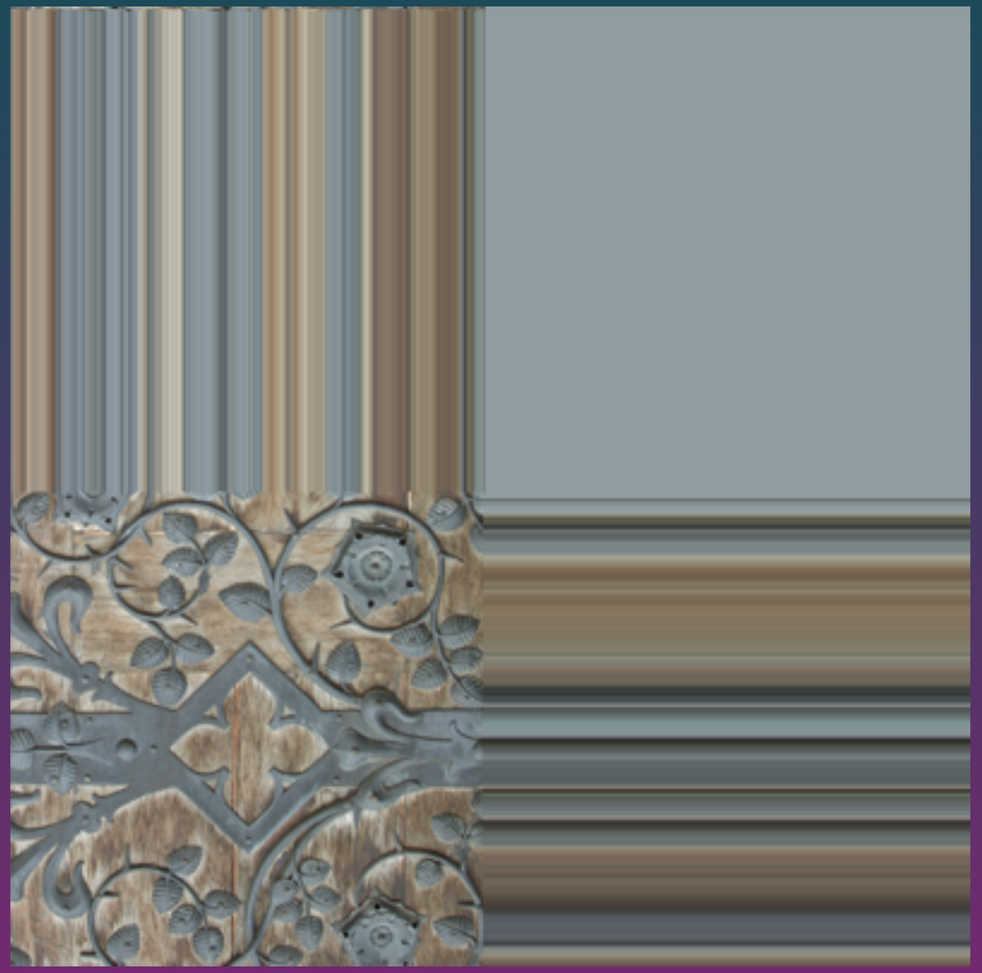
0,0

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.

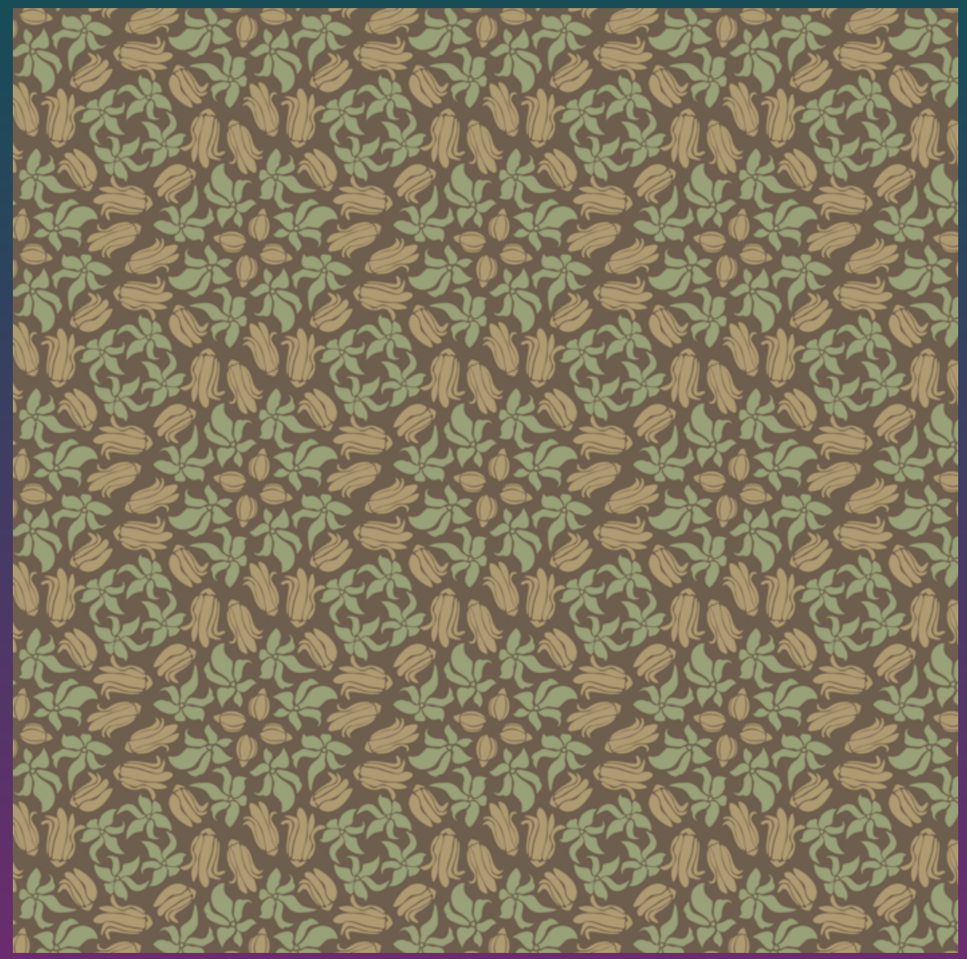
CLAMPING

```
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 tiled textures like patterns, etc.



Use GL_CLAMP for non-tiled images with alpha.