Blending, sampling and aliasing.

CS GY-6533 / UY-4533

Blending

Enabling blending.

```
glEnable(GL_BLEND);
```

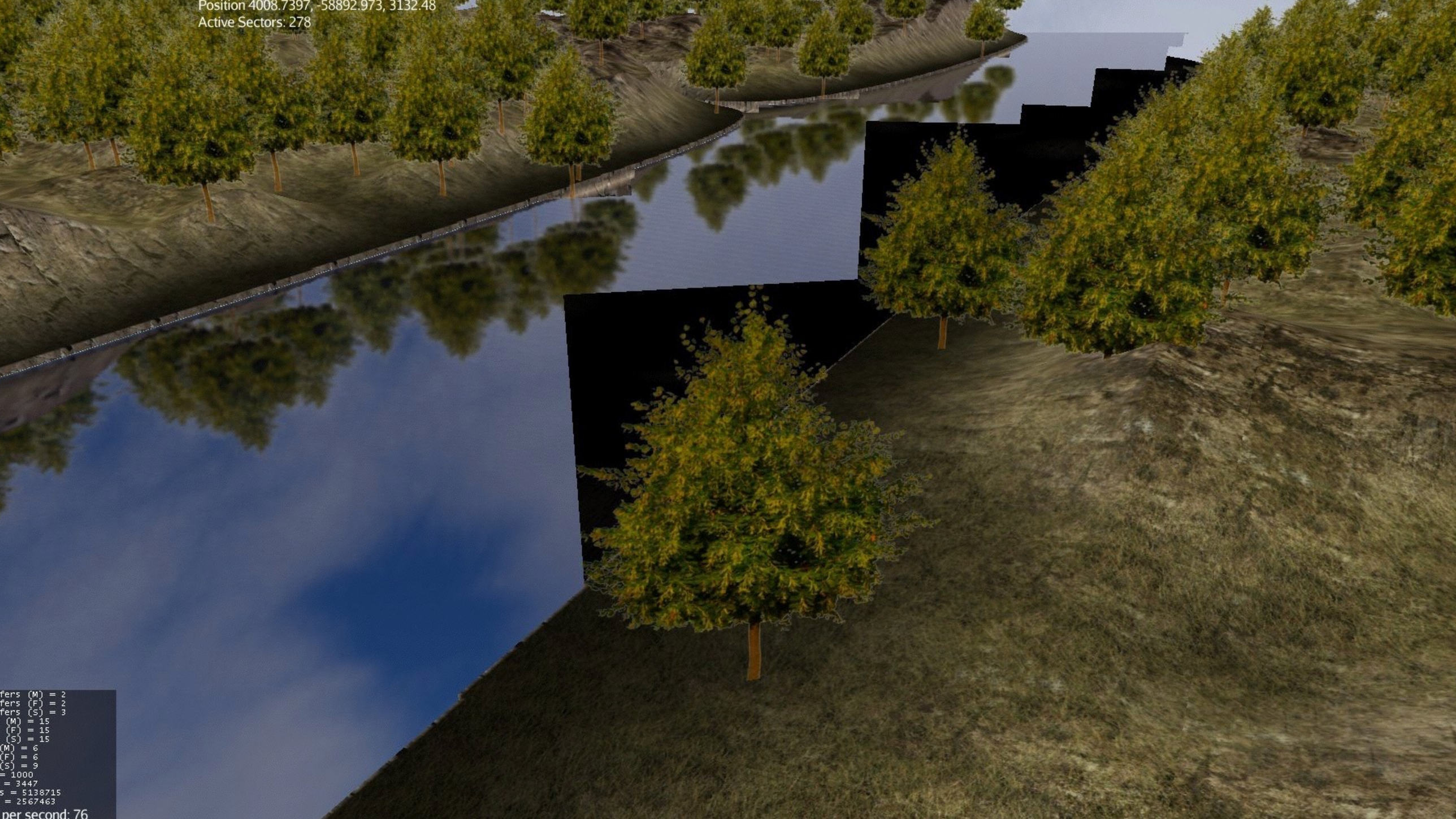
Alpha blending function.

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Additive blending function.

```
glBlendFunc (GL_SRC_ALPHA, GL_ONE);
```

Blending issues with the z-buffer.


Position 4008.7397, -58892.973, 3132.48
Active Sectors: 278
fers (M) = 2
fers (F) = 2
fers (S) = 3
(M) = 15
(F) = 15
(S) = 15
(M) = 6
(F) = 6
(S) = 9
= 1000
= 3447
s = 5138715
= 2567463
ner second: 76

Order of drawing matters when blending because blending happens between the pixel that is already on the screen and the pixel that is about to be drawn.

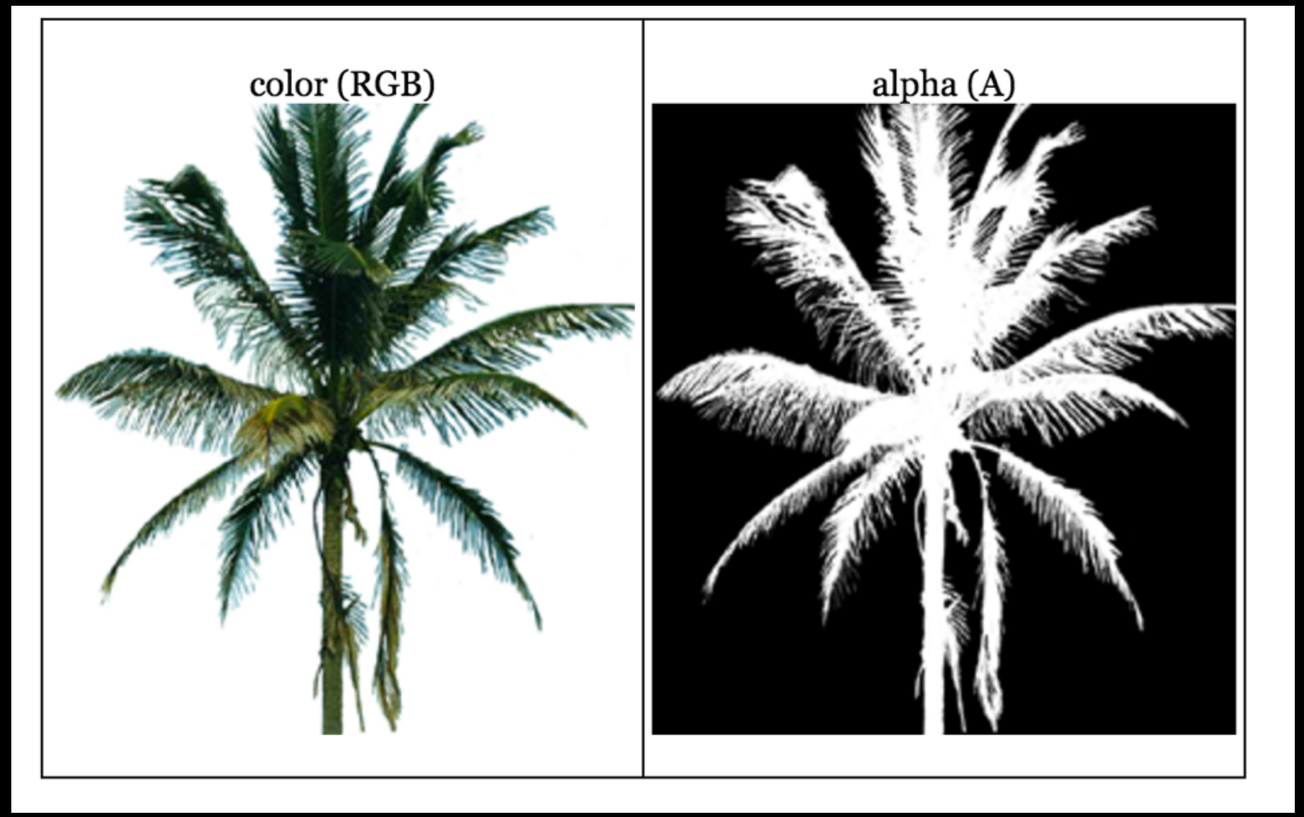
Discarding fragments.

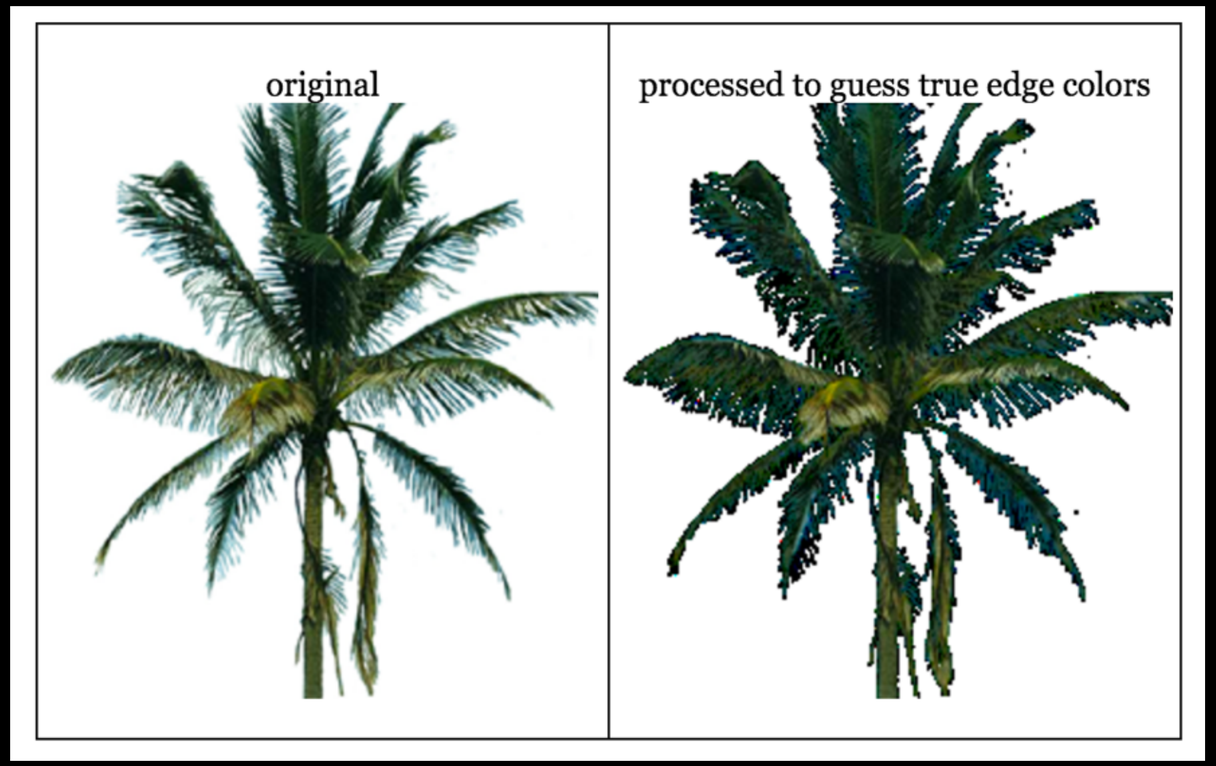
Discard fragment if alpha value is less than 1.0

```
uniform sampler2D diffuse;
varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D(diffuse, texCoordVar);
    if(gl_FragColor.a < 1.0) {
        discard;
    }
}
```

Discarded fragments are not written to the color or depth buffers.

Premultiplied alpha.


color (RGB)
alpha (A)


original
processed to guess true edge colors

Alpha pre-multiplication must be done manually.

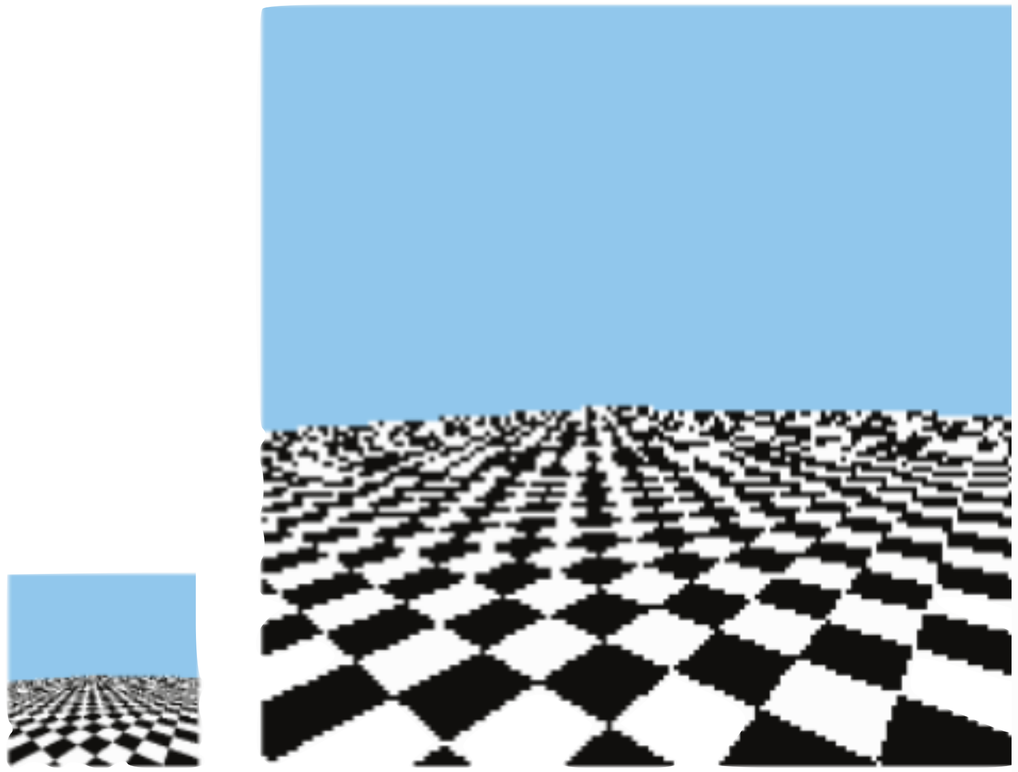
We must also use a slightly different blending function.

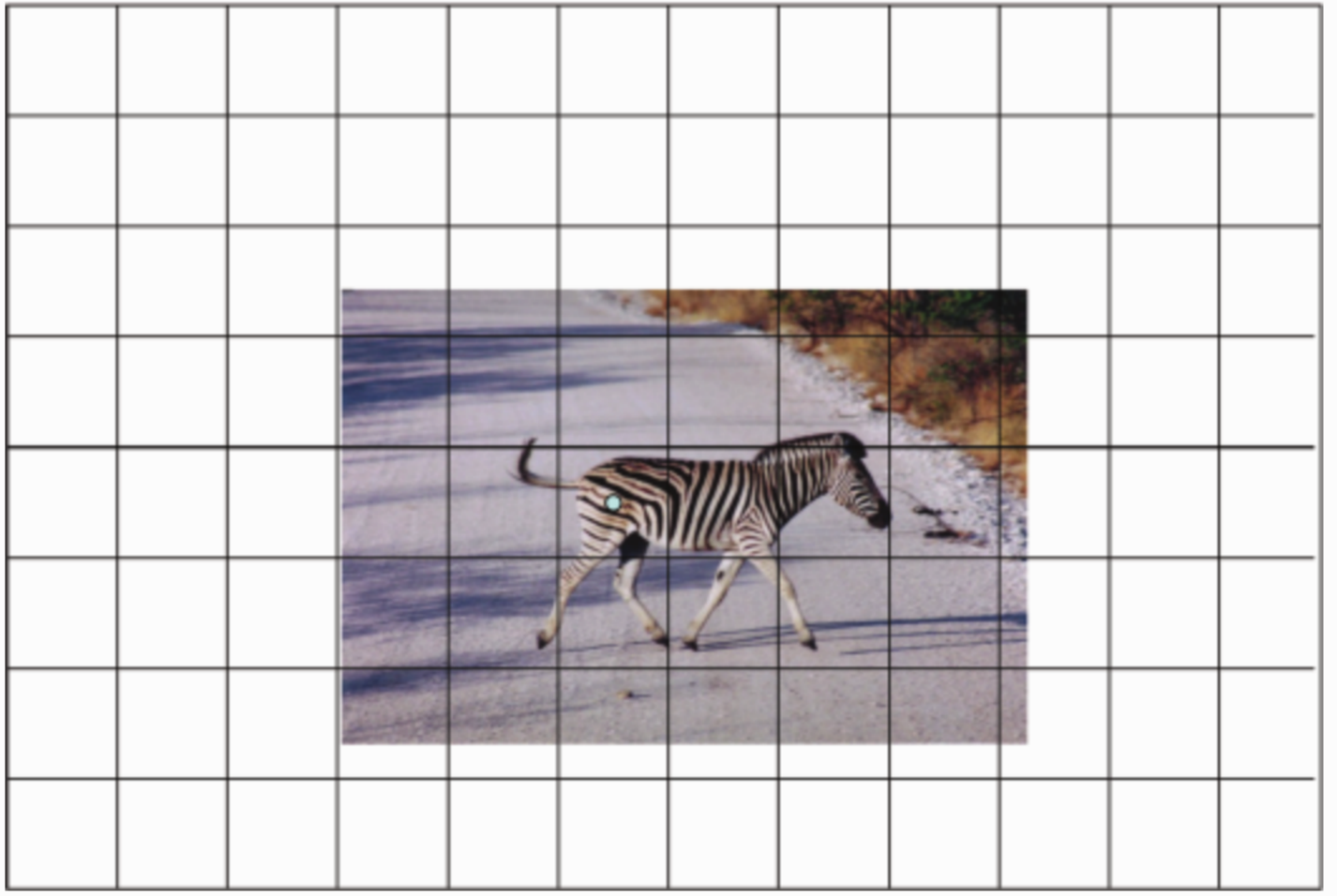
```
glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
```

Sampling.

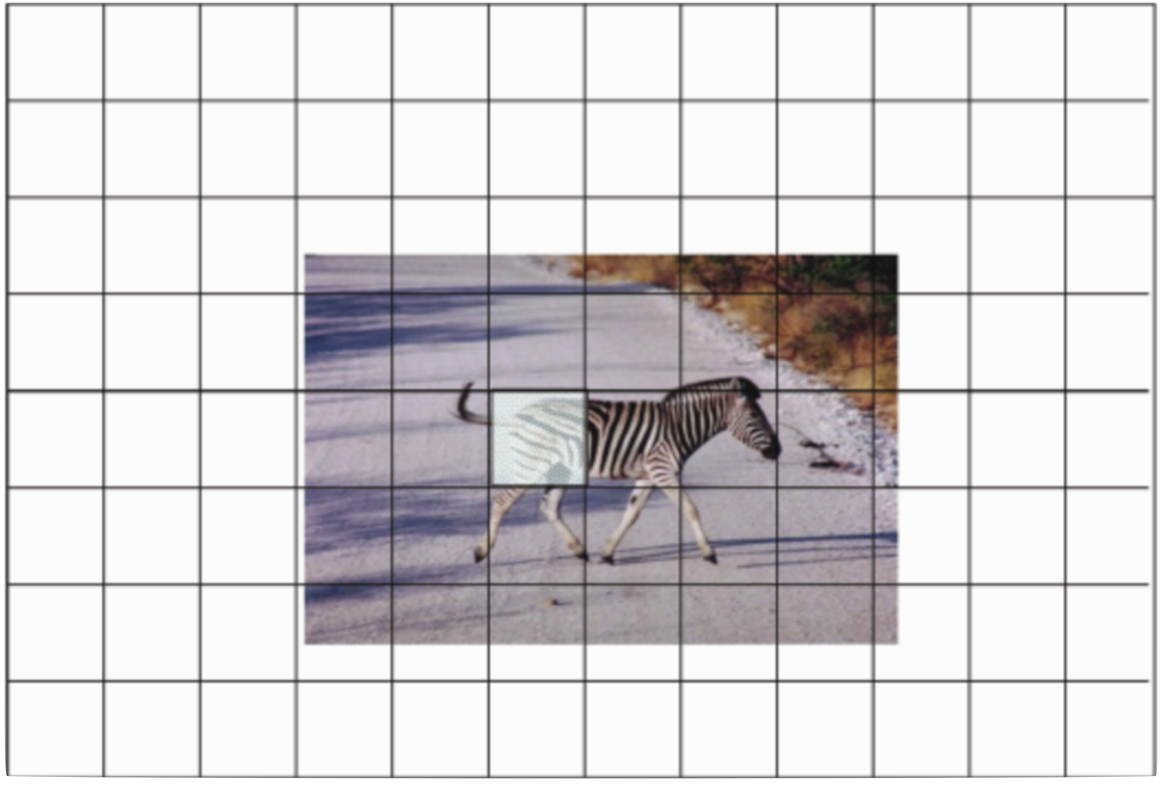
Continuous and discrete images.

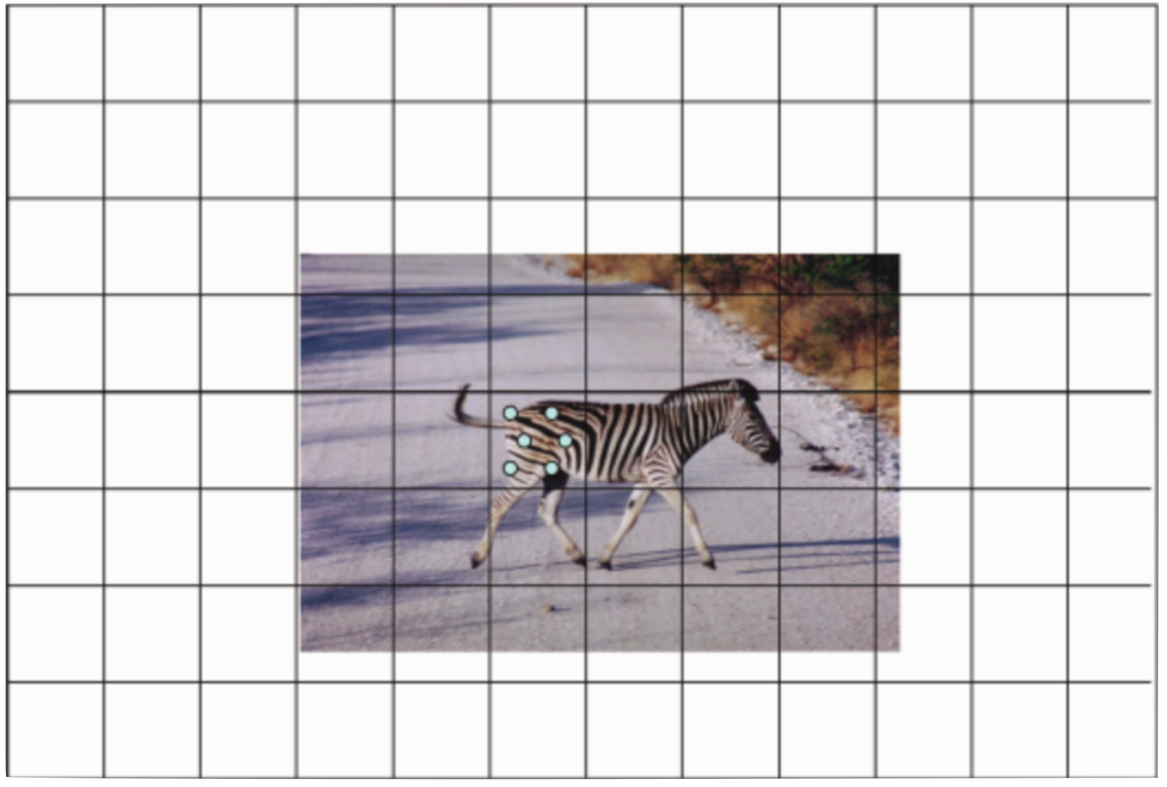
$$\texttt{I[i][j]} \leftarrow I(i, j).$$



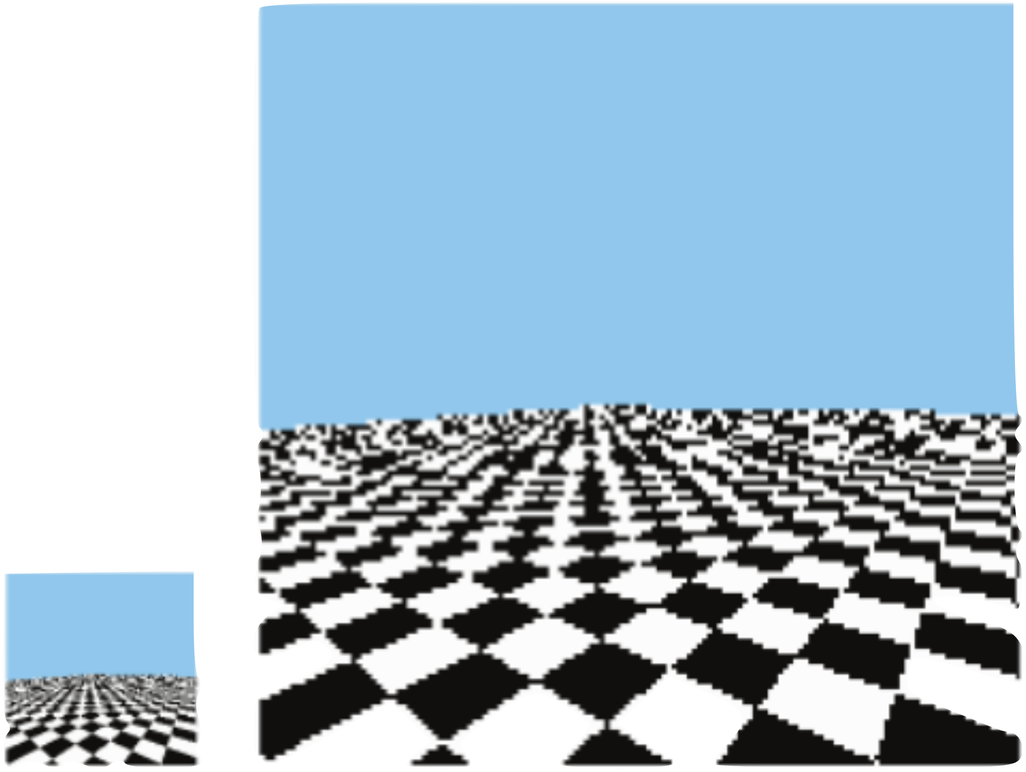


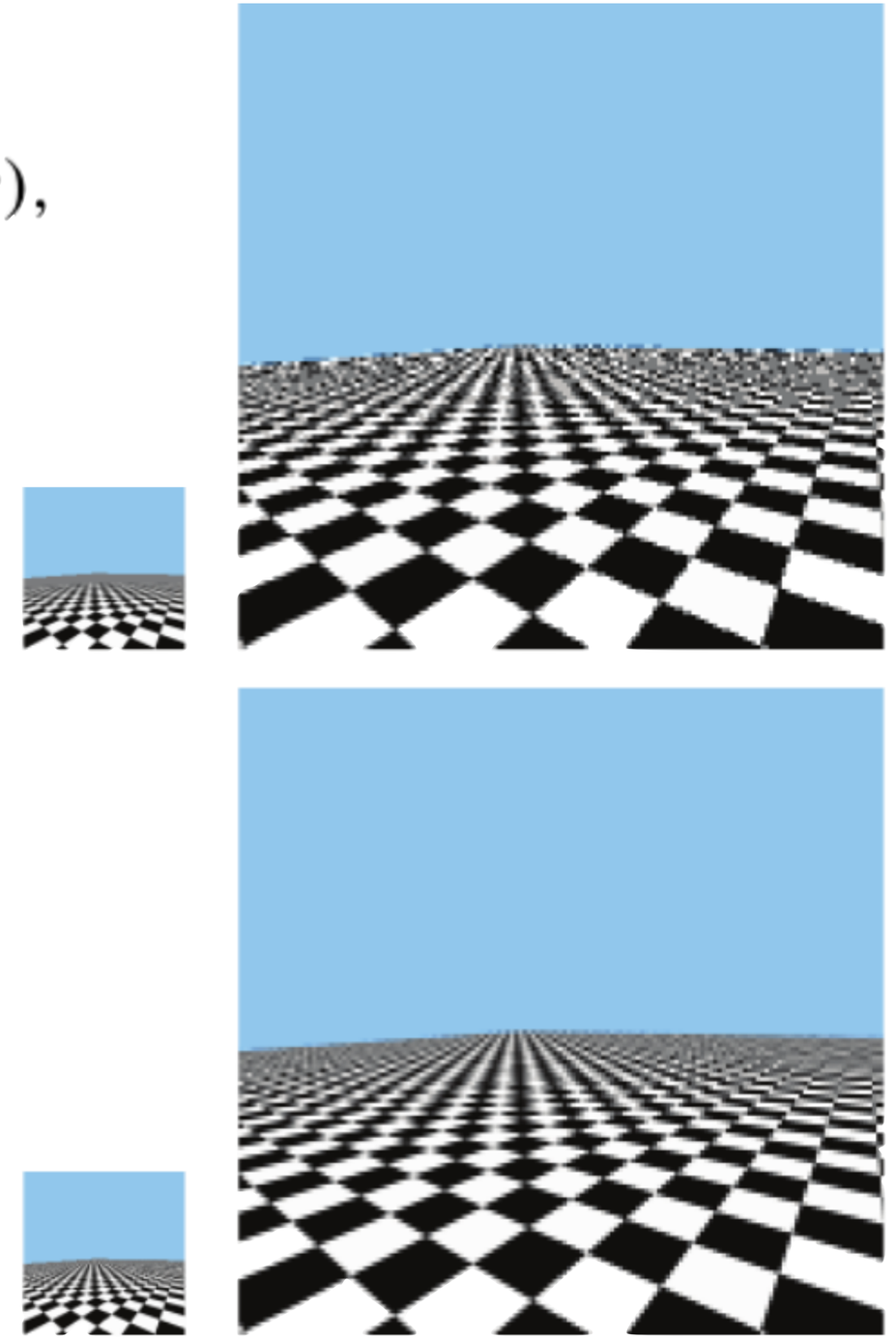
$$\texttt{I[i][j]} \leftarrow \int\int_{\Omega} dx\, dy\; I(x, y) F_{i,j}(x, y),$$





$$\texttt{I[i][j]} \leftarrow \int\int_{\Omega} dx\, dy\, I(x, y) F_{i,j}(x, y),$$





Reconstruction.

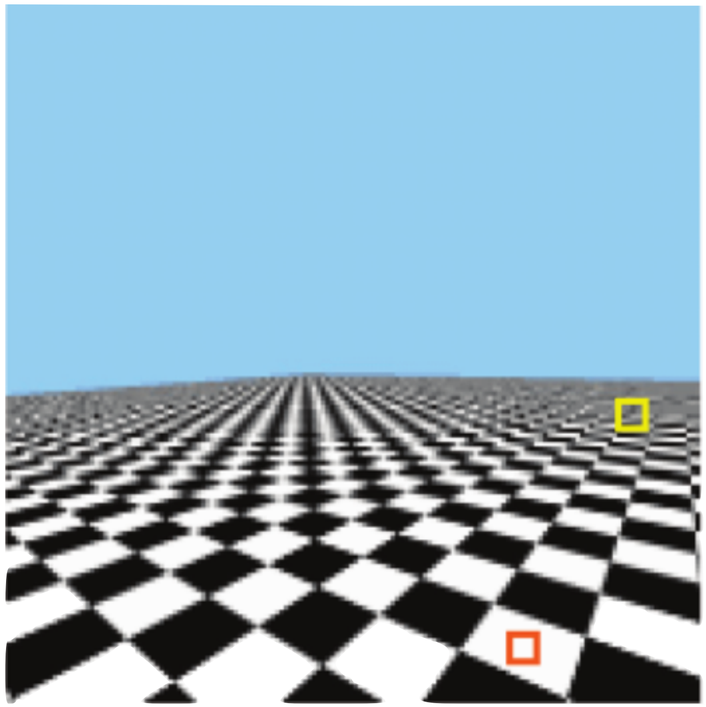
Linear reconstruction.

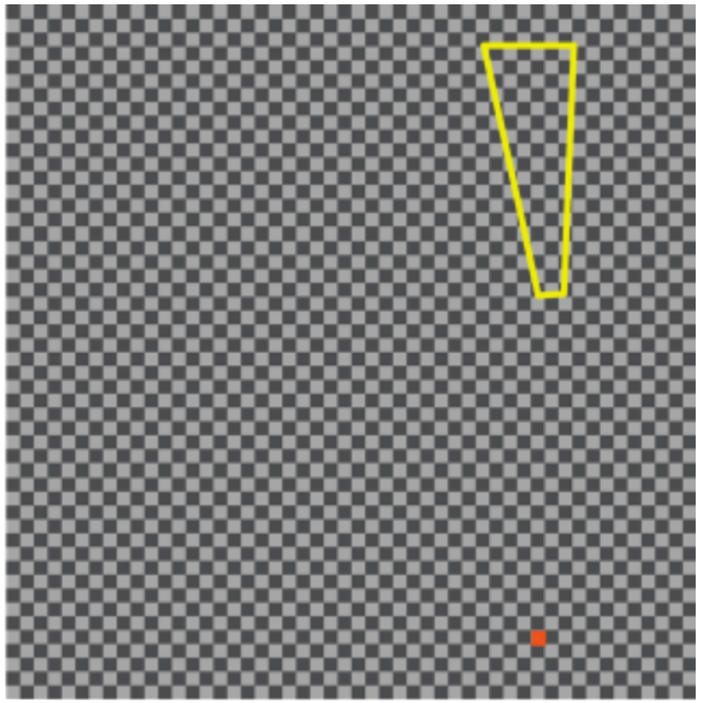
```
color constantReconstruction(float x, float y, color image[][]) {
  int i = (int) (x + .5);
  int j = (int) (y + .5);
  return image[i][j]
}
```

Bilinear reconstruction.

```
color bilinearReconstruction(float x, float y, color image[][]) {

  int intx = (int) x;
  int inty = (int) y;

  float fracx = x - intx;
  float fracy = y - inty;

  color colorx1 = (1-fracx)* image[intx] [inty] + (fracx) * image[intx+1][inty];
  color colorx2 = (1-fracx)* image[intx] [inty+1] + (fracx) * image[intx+1][ inty+1];
  color colorxy = (1-fracy)* colorx1 + (fracy) * colorx2;

  return(colorxy)
}
```

Resampling.





Setting texture filtering modes.

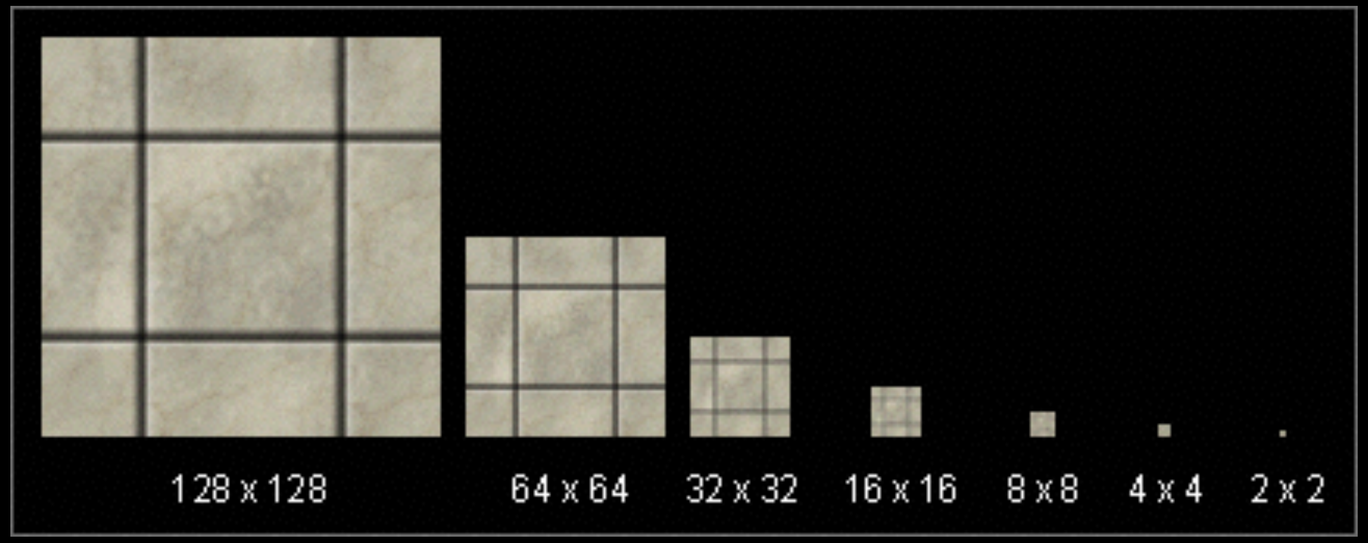
Constant.

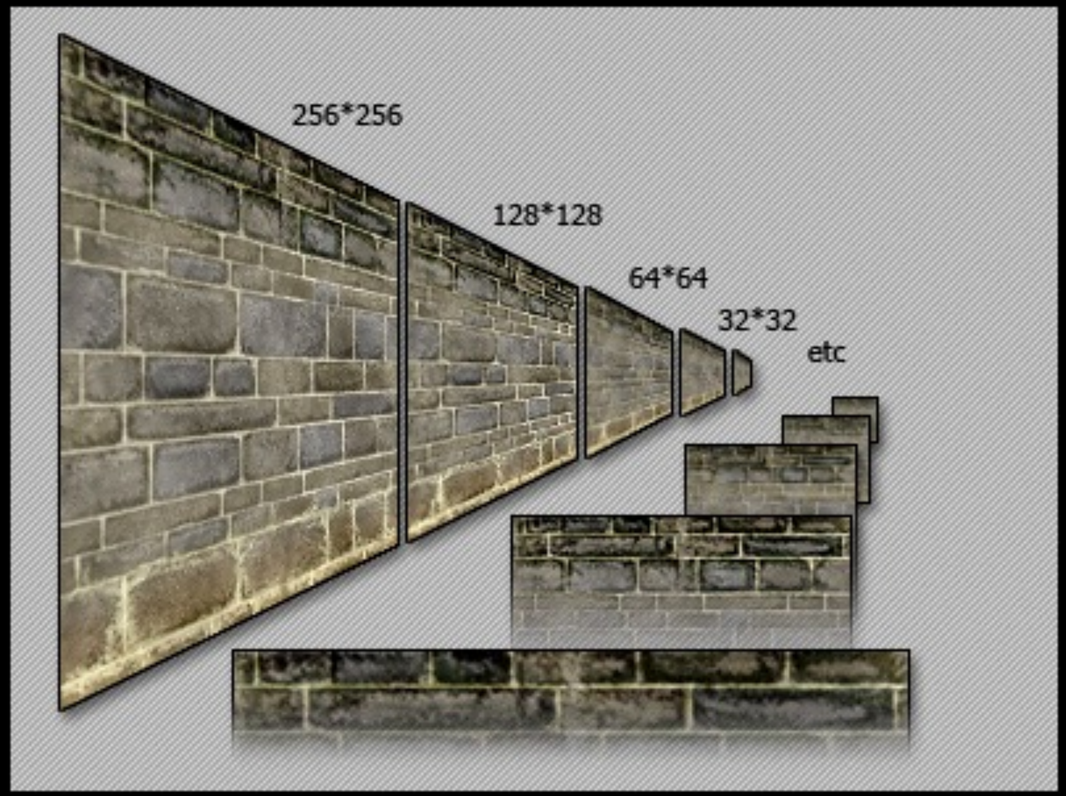
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

Bilinear.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

Mip-mapping.


128 x 128
64 x 64
32 x 32
16 x 16
8 x 8
4 x 4
2 x 2


256*256
128*128
64*64
32*32
etc

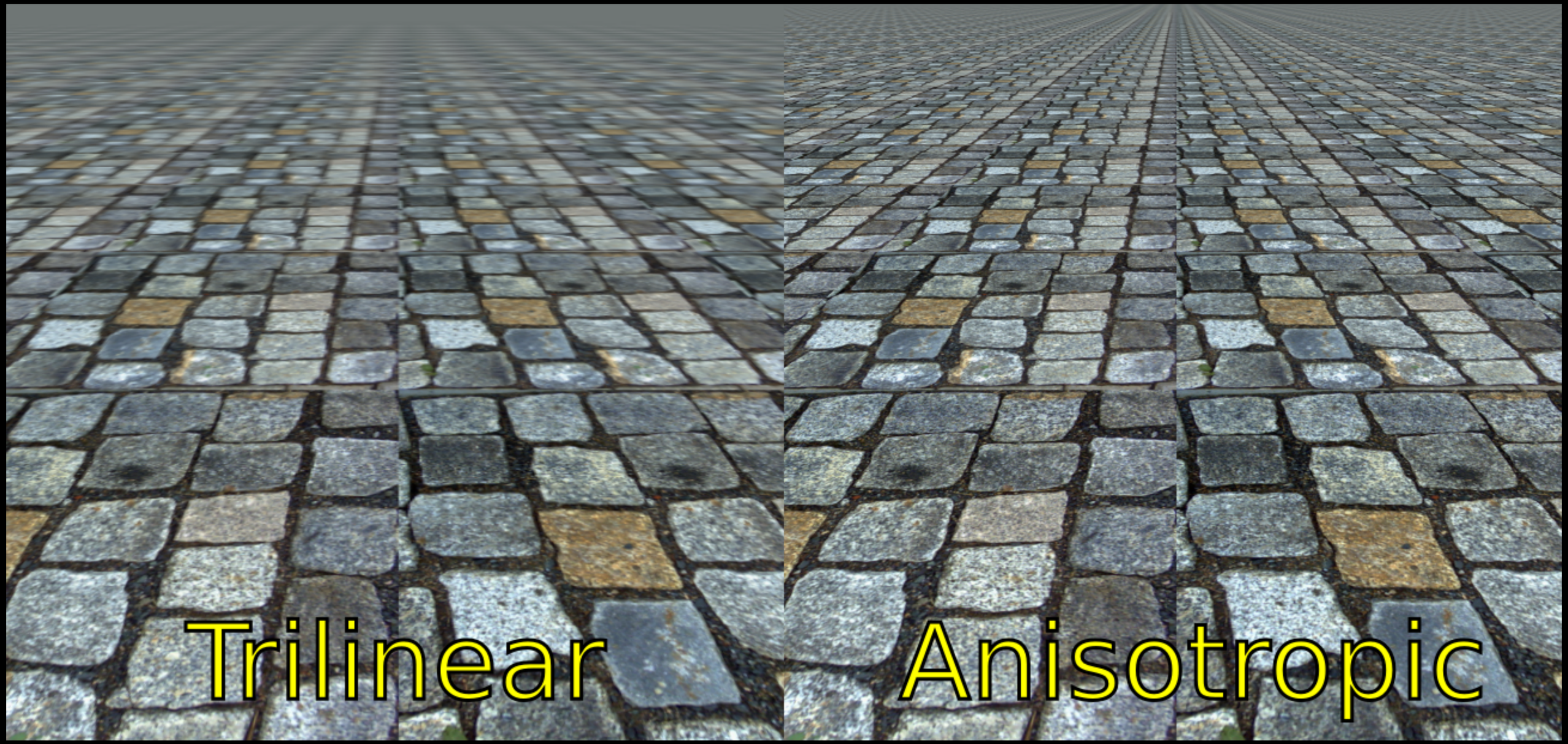
Setting mipmap filter.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
```

Creating the mipmaps.

```
glGenerateMipmap(GL_TEXTURE_2D);
```

Anisotropic filtering.


Trilinear
Anisotropic

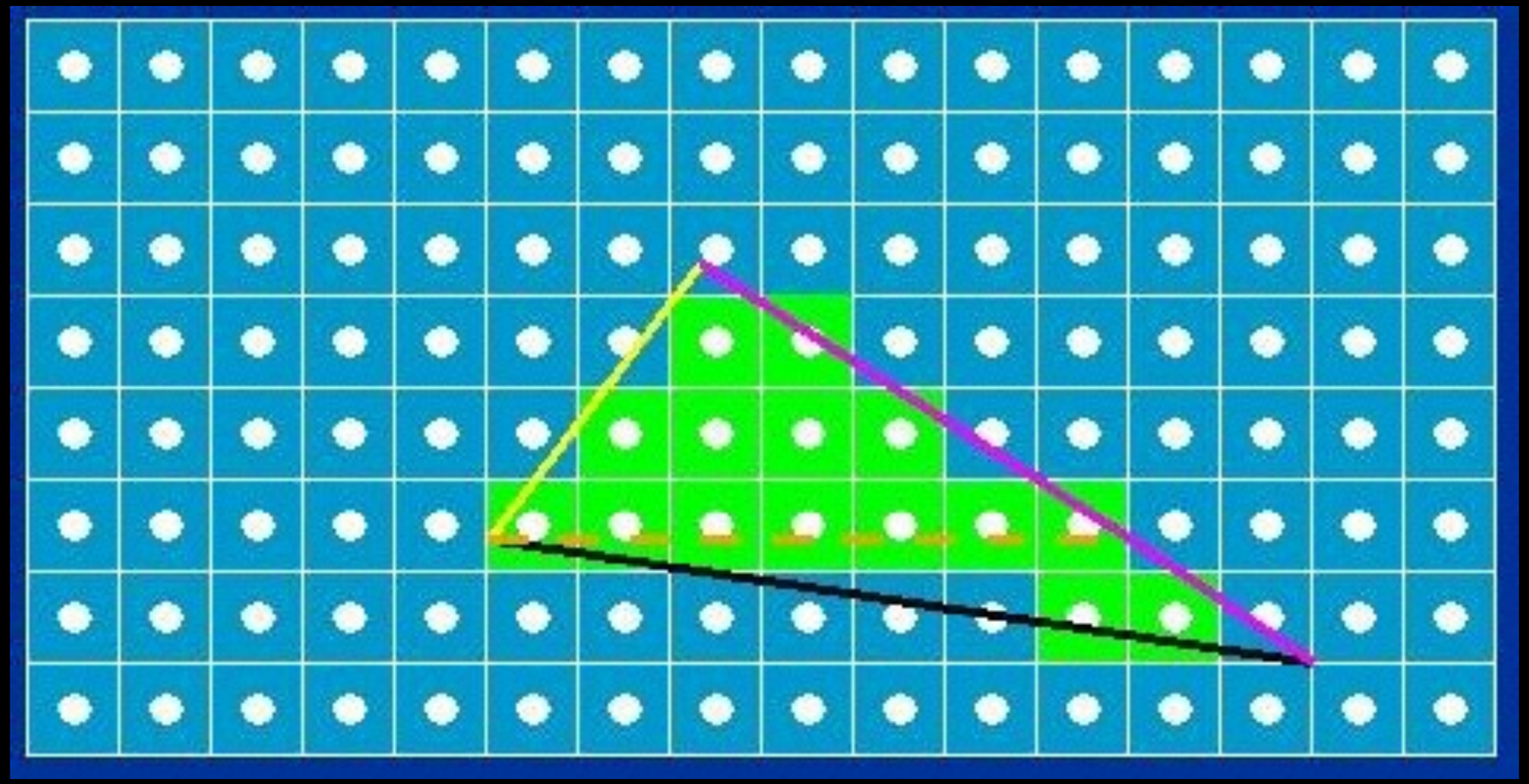
Enabling anisotropic filtering in OpenGL.

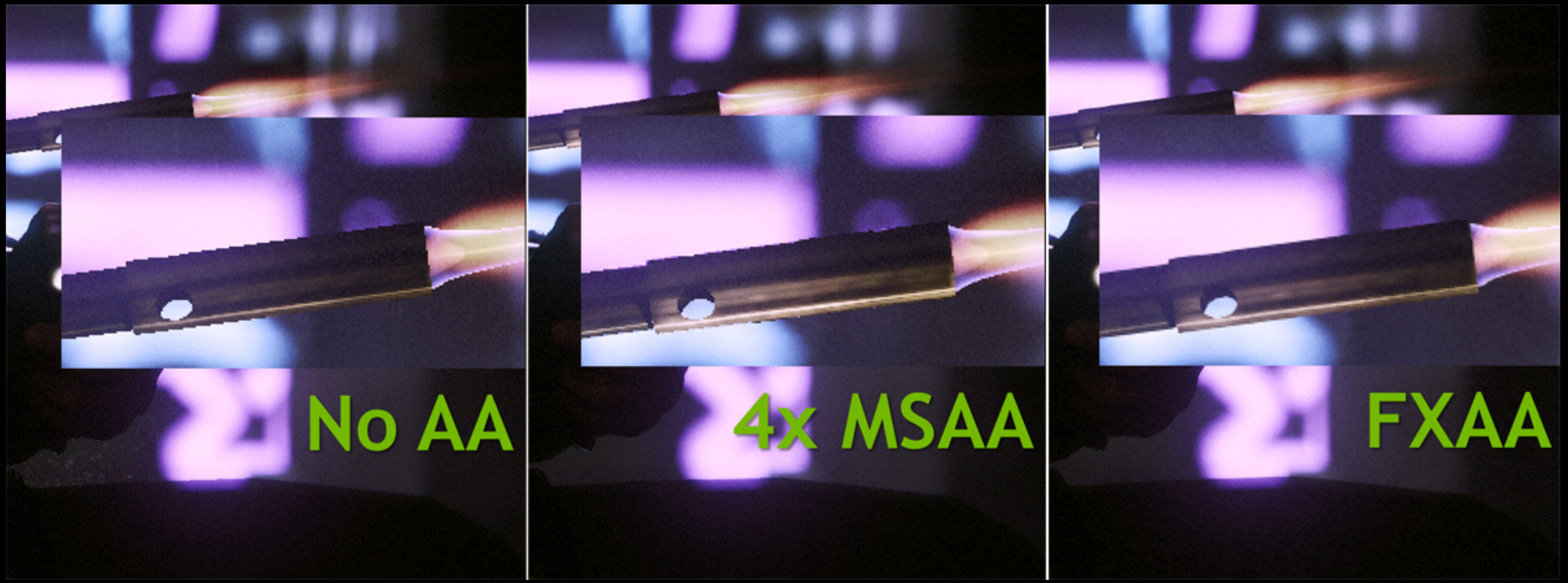
```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, anisotropyAmount);
```

The range of anisotropy amount can be gotten using the following code (generally the values are 2,4,8 and 16).

```
GLfloat fLargest;
glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &fLargest);
```

Edge aliasing.




No AA
4x MSAA
FXAA

MSAA (Multi Sampling Anti Aliasing)

Enabling MSAA in GLUT.

```
glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH | GLUT_MULTISAMPLE );

glEnable(GL_MULTISAMPLE);
```

FXAA (Fast Approximate Anti Aliasing)


No AA
FXAA

Writing post-processing shaders.

- 1. Render your scene to texture.
- 2. Display the texture on screen using two triangles.

Vertex shader.

```
attribute vec4 position;
attribute vec2 texCoord;

varying vec2 texCoordVar;
void main()
{
    gl_Position =  position;
    texCoordVar = texCoord;
}
```

No need for projection or modelview matrices as we are just drawing two fullscreen triangles.

Fragment shader.

```
uniform sampler2D screenFramebuffer;
varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D( screenFramebuffer, texCoordVar);
}
```

Just draw the texture as is.

Create your two triangle buffers.

```
GLfloat screenTriangleUVs[] = {
    1.0f, 1.0f,
    1.0f, 0.0f,
    0.0f, 0.0,

    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 1.0f
};

GLfloat screenTrianglePositions[] = {
    1.0f, 1.0f,
    1.0f, -1.0f,
    -1.0f, -1.0f,

    -1.0f, -1.0f,
    -1.0f, 1.0f,
    1.0f, 1.0f
};
```

After drawing your scene to the framebuffer,
unbind it and draw your two triangles using the screen shader
with frame buffer texture as the texture.

```
glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer);

// draw our scene

glBindFramebuffer(GL_FRAMEBUFFER, 0);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

glUseProgram(screenTrianglesProgram);

glUniform1i(screenFramebufferUniform, 0);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, frameBufferTexture);

glBindBuffer(GL_ARRAY_BUFFER, screenTrianglesPositionBuffer);
glVertexAttribPointer(screenTrianglesPositionAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(screenTrianglesPositionAttribute);

glBindBuffer(GL_ARRAY_BUFFER, screenTrianglesUVBuffer);
glVertexAttribPointer(screenTrianglesTexCoordAttribute, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(screenTrianglesTexCoordAttribute);

glDrawArrays(GL_TRIANGLES, 0, 6);
glDisableVertexAttribArray(screenTrianglesPositionAttribute);
glDisableVertexAttribArray(screenTrianglesTexCoordAttribute);
```

Basic screen shader examples.

Inverting screen colors.

```
uniform sampler2D texture;
varying vec2 texCoordVar;

void main()
{
    gl_FragColor = vec4(1.0-texture2D( texture, texCoordVar).xyz, 1.0);
}
```

Black and white.

```
uniform sampler2D texture;
varying vec2 texCoordVar;

void main()
{
    vec4 texColor = texture2D( texture, texCoordVar);
    float brightness = (texColor.x+texColor.y+texColor.z)/3.0;
    gl_FragColor = vec4(brightness, brightness, brightness, 1.0);
}
```

The FXAA shader.

Need to pass screen resolution to the shader if we want to deal with pixels (fragment shader has no concept of a pixel size).

```
uniform sampler2D screenFramebuffer;
varying vec2 texCoordVar;

uniform vec2 resolution;

void main()
{

    vec2 v_rgbNW;
    vec2 v_rgbNE;
    vec2 v_rgbSW;
    vec2 v_rgbSE;
    vec2 v_rgbM;

    texcoords(texCoordVar * resolution, resolution, v_rgbNW, v_rgbNE, v_rgbSW, v_rgbSE, v_rgbM);

    gl_FragColor = fxaa(screenFramebuffer, texCoordVar * resolution, resolution, v_rgbNW,
v_rgbNE, v_rgbSW, v_rgbSE, v_rgbM);

}
```

```
void texcoords(vec2 fragCoord, vec2 resolution,
               out vec2 v_rgbNW, out vec2 v_rgbNE,
               out vec2 v_rgbSW, out vec2 v_rgbSE,
               out vec2 v_rgbM) {

    vec2 inverseVP = 1.0 / resolution.xy;
    v_rgbNW = (fragCoord + vec2(-1.0, -1.0)) * inverseVP;
    v_rgbNE = (fragCoord + vec2(1.0, -1.0)) * inverseVP;
    v_rgbSW = (fragCoord + vec2(-1.0, 1.0)) * inverseVP;
    v_rgbSE = (fragCoord + vec2(1.0, 1.0)) * inverseVP;
    v_rgbM = vec2(fragCoord * inverseVP);

}
```

```
#define FXAA_REDUCE_MIN   (1.0/ 128.0)
#define FXAA_REDUCE_MUL   (1.0 / 8.0)
#define FXAA_SPAN_MAX     8.0

vec4 fxaa(sampler2D tex, vec2 fragCoord, vec2 resolution,
          vec2 v_rgbNW, vec2 v_rgbNE,
          vec2 v_rgbSW, vec2 v_rgbSE,
          vec2 v_rgbM) {
    vec4 color;
    vec2 inverseVP = vec2(1.0 / resolution.x, 1.0 / resolution.y);
    vec3 rgbNW = texture2D(tex, v_rgbNW).xyz;
    vec3 rgbNE = texture2D(tex, v_rgbNE).xyz;
    vec3 rgbSW = texture2D(tex, v_rgbSW).xyz;
    vec3 rgbSE = texture2D(tex, v_rgbSE).xyz;
    vec4 texColor = texture2D(tex, v_rgbM);
    vec3 rgbM  = texColor.xyz;
    vec3 luma = vec3(0.299, 0.587, 0.114);
    float lumaNW = dot(rgbNW, luma);
    float lumaNE = dot(rgbNE, luma);
    float lumaSW = dot(rgbSW, luma);
    float lumaSE = dot(rgbSE, luma);
    float lumaM  = dot(rgbM,  luma);
    float lumaMin = min(lumaM, min(min(lumaNW, lumaNE), min(lumaSW, lumaSE)));
    float lumaMax = max(lumaM, max(max(lumaNW, lumaNE), max(lumaSW, lumaSE)));

    vec2 dir;
    dir.x = -((lumaNW + lumaNE) - (lumaSW + lumaSE));
    dir.y =  ((lumaNW + lumaSW) - (lumaNE + lumaSE));

    float dirReduce = max((lumaNW + lumaNE + lumaSW + lumaSE) *
                          (0.25 * FXAA_REDUCE_MUL), FXAA_REDUCE_MIN);

    float rcpDirMin = 1.0 / (min(abs(dir.x), abs(dir.y)) + dirReduce);
    dir = min(vec2(FXAA_SPAN_MAX, FXAA_SPAN_MAX),
              max(vec2(-FXAA_SPAN_MAX, -FXAA_SPAN_MAX),
                  dir * rcpDirMin)) * inverseVP;

    vec3 rgbA = 0.5 * (
                       texture2D(tex, fragCoord * inverseVP + dir * (1.0 / 3.0 - 0.5)).xyz +
                       texture2D(tex, fragCoord * inverseVP + dir * (2.0 / 3.0 - 0.5)).xyz);
    vec3 rgbB = rgbA * 0.5 + 0.25 * (
                                     texture2D(tex, fragCoord * inverseVP + dir * -0.5).xyz +
                                     texture2D(tex, fragCoord * inverseVP + dir * 0.5).xyz);

    float lumaB = dot(rgbB, luma);
    if ((lumaB < lumaMin) || (lumaB > lumaMax))
        color = vec4(rgbA, texColor.a);
    else
        color = vec4(rgbB, texColor.a);
    return color;
}
```

NO FXAA

WITH FXAA