At each rendered pixel, color is calculated by selected texels: nearest texel, bilinear interpolation, bi-cubic interpolation, etc.

The calculated color then substitute for the pixel color or scale one or more of the surface’s material properties, such as its diffuse color components.
An Overview

• Steps in Texture Mapping

1) Specify the texture. (R, G, B, A, mipmapping)
2) Indicate how the texture is to be applied to each pixel. (decal, modulate, blend)
3) Enable texture mapping. (glEnable()
   GL_TEXTURE_1D or GL_TEXTURE_2D
4) Draw the scene, supplying both texture and geometric coordinates.
• Texture mapping works only in RGB mode in OpenGL

• Example: A Texture-Mapped Checkerboard: checker.c

```c
void makeCheckImage(void)
{
    int i, j, c;
    for (i = 0; i < checkImageHeight; i++) {
        for (j = 0; j < checkImageWidth; j++) {
            c = ((((i&0x8)==0)^((j&0x8))==0))*255;
            checkImage[i][j][0] = (GLubyte) c;
            checkImage[i][j][1] = (GLubyte) c;
            checkImage[i][j][2] = (GLubyte) c;
            checkImage[i][j][3] = (GLubyte) 255;
        }
    }
}
```

void init(void){
    makeCheckImage();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

    glTexImage2D(GL_TEXTURE_2D, 0, 4, checkImageWidth, checkImageHeight,
        0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
}
void display(void)
{
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(1.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
    glEnd();
    glFlush();
    glDisable(GL_TEXTURE_2D);
}
Pixel Calculation

- A pixel may be a portion of a texel (Magnification)
- A pixel may cover an area of texels (Minification)
- Many solutions (for both magni. & mini.)
  - Nearest texel for the pixel
  - Linear interpolation of surrounding texels for the pixel
Texture Interpolation

- Specify where the vertices in world space are mapped to in texture space
  - $s = s(x, y, z)$
  - $t = t(x, y, z)$
  - Straight lines in world space go to straight lines in texture space
Interpolating texture coordinates to find a pixel color

Pixel color depends on texture coord. and the texels at the coord.
Nearest texel or Bi-linear Interpolation
(Minification or Magnification)

\[
P(2,1+d) = P(2,1)*(1-d) + P(2,2)*d
\]

\[
P(1, 1+d) = P(1,1)*(1-d) + P(1,2)*d
\]

\[
P' = P(1,1+d)*(1-d') + P(2,1+d)*d'
\]
Transform texture coordinates

- All the texture coordinates are multiplied by **GL_TEXTURE matrix** before being used

- To transform texture coordinates, you do:
  - `glMatrixMode(GL_TEXTURE);`
  - `glTexCoord` specifies texture coordinates in one, two, three, or four dimensions.
    - `glTexCoord1` sets the current texture coordinates to (s, 0, 0, 1)
    - `glTexCoord2` sets them to (s, t, 0, 1);
    - `glTexCoord3` sets them to (s, t, r, 1);
    - `glTexCoord4` sets them to (s, t, r, q);
Texture mapping is done as the primitives are rasterized:

- Given pixel coordinates, calculate corresponding texture coordinates
- Texture coordinates are transformed by its matrix
- Pixel color is calculated according to texels surround the transformed texture coordinates
void initTexture(void)
{
    read_stars_image();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

    // glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    // glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_LUMINANCE, stars_pixels, stars_pixels,
                 0, GL_LUMINANCE, GL_UNSIGNED_BYTE, stars_image);
}
void drawTexture(float x, float y, float z)
{
   //   glRasterPos3f(x, y, z);

   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

   glEnable(GL_TEXTURE_2D);

   glBegin(GL_QUADS);
      glTexCoord2f(0.0, 0.0); glVertex3f(x, y, z);
      glTexCoord2f(0.0, 1.0); glVertex3f(-x, y, z);
      glTexCoord2f(1.0, 1.0); glVertex3f(-x, -y, z);
      glTexCoord2f(1.0, 0.0); glVertex3f(x, -y, z);
   glEnd();
   glDisable(GL_TEXTURE_2D);
}
void display(void)
{
    drawTexture(-2.4*Width, -2.4*Height, -1.9*Width);

    drawRobot(A, B, C, alpha, beta, gama);

    glutSwapBuffers();
}
Specifying the Texture

```c
void glTexImage2D(GLenum target, GLint level,
    GLint components, GLsizei width, GLsizei height,
    GLint border, GLenum format, GLenum type, const GLvoid *pixels);
```

1) The `target` parameter: constant `GL_TEXTURE_2D`.
2) The `level` parameter for multiple resolutions
3) `Components` is 1 of 38 symbolic constants. A value of 1 selects the R component, 2 selects the R and A components, 3 selects R, G, and B, and 4 selects R, B, G, and A. It indicates which values are selected for texels.
4) The `width` and `height` parameters give the dimensions of the texture image.
5) The `border` indicates the width of the border, which is usually zero.
6) The `format` and `type` parameters describe the format and data type of the texture image data. (GL_COLOR_INDEX, GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_LUMINANCE, or GL_LUMINANCE_ALPHA, corresponding to 3)
Some Minor Things

1. The number of texels for both the width and height of a texture image, not including the optional border, must be a **power of 2**.

2. `gluScaleImage()` correct/alter the sizes of your textures

3. `glCopyTexImage2D()` creates a 2D texture using framebuffer data

1D textures and 3D textures

- 3D textures are used for rendering in medical and geoscience applications (CT or MRI images)

  ```
  void glTexImage1D()
  void glTexImage3D()
  ```
Pixel and Texel Relations

- Corresponding vertices of texture and polygons

```c
glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(x, y, z);
    glTexCoord2f(0.0, 1.0); glVertex3f(-x, y, z);
    glTexCoord2f(1.0, 1.0); glVertex3f(-x, -y, z);
    glTexCoord2f(1.0, 0.0); glVertex3f(x, -y, z);
```

- Antialiasing
  - Pre-filtering: Filter the texture down before applying it (mipmaping)
  - Post-filtering: Take multiple texels from the texture and filter them before applying to the polygon fragment (linear interpolation)

- Texel color components replace or modulate pixel color
  - GL_MODULATE – multiply texture and object color
  - GL_BLEND – linear combination of texture and object color
  - GL_REPLACE – use texture color to replace object color
Multiple Levels of Detail (Pre-filtering)

- textures that are mapped onto smaller objects might shimmer and flash as the objects move.
- To use mipmapping, you provide all sizes of your texture in powers of 2 between the largest size and a 1x1 map.
- OpenGL chooses matching size for texture mapping
A Mipmapping Example: J4_8_Mipmap

A Mipmapping Example: mmap.c

```c
void myinit(void) {
    ...
    loadImages();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, 32, 32, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage32[0][0][0]);
    glTexImage2D(GL_TEXTURE_2D, 1, 3, 16, 16, 0, GL_RGB, GL_UNSIGNED_BYTE, &mipmapImage16[0][0][0])
    ...
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
}
```
Mipmap Math

• Define a scale factor, $\rho = \text{texels/pixel}$
  – It can be derived from the transformation matrices
• Define $\lambda = \log_2 \rho$ ($\rho = 2^\lambda$)
• $\lambda$ tells you which mipmap level to use
  – Level 0 is the original texture
  – Level 1 is the next smallest texture, and so on
  – If $\lambda < 0$, then multiple pixels map to one texel: no mipmap consideration necessary (magnification)
Post-Filtering

• Magnification: When $\lambda < 0$ the image pixel is smaller than the texel:
  – `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, type)`
  – Type is `GL_LINEAR` or `GL_NEAREST`

• Minification: When $\lambda > 0$ the image pixel is bigger than the texel:
  – `GL_TEX_MIN_FILTER`
  – Can choose to:
    • Take nearest point in base texture, `GL_NEAREST`
    • Linearly interpolate nearest 4 texels in base texture, `GL_LINEAR`
    • Take the nearest mipmap and then take the nearest texel or interpolate in that mipmap, `GL_XXXX_MIPMAP_NEAREST`
    • Interpolate between the two nearest mipmaps using nearest or interpolated points from each, `GL_XXXX_MIPMAP_LINEAR`
glTexParameteri(GL_TEXTURE_2D
    GL_TEXTURE_MAG_FILTER,  GL_NEAREST);

```c
    GL_TEXTURE_MIN_FILTER,  GL_NEAREST);
```  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_TEXTURE_MAG_FILTER</td>
<td>GL_NEAREST or GL_LINEAR</td>
</tr>
<tr>
<td>GL_TEXTURE_MIN_FILTER</td>
<td>GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, or GL_LINEAR_MIPMAP_LINEAR</td>
</tr>
</tbody>
</table>

Copyright © 2002 by Jim X. Chen: jchen@cs.gmu.edu
• GL_NEAREST: no filtering (interpolation), no mipmap
• GL_LINEAR: filtering, no mipmap
• GL_NEAREST_MIPMAP_NEAREST: no filtering, sharp switching between mipmaps; nearest texel in a nearest mipmap
• GL_NEAREST_MIPMAP_LINEAR - no filtering, smooth transition between mipmaps; linear interpolation between 2 texels in 2 mipmaps
• GL_LINEAR_MIPMAP_NEAREST - filtering, sharp switching between mipmaps; bi-linear interpolation within a mipmap.
• GL_LINEAR_MIPMAP_LINEAR - filtering, smooth transition between mipmaps; tri-linear interpolation
• So:
  GL_LINEAR is bilinear
  GL_LINEAR_MIPMAP_NEAREST is bilinear with a chosen mipmap
  GL_LINEAR_MIPMAP_LINEAR is trilinear between two mipmaps
Trilinear filtering
(GL_LINEAR_MIPMAP_LINEAR)

- Trilinear filtering is an extension of the bilinear texture filtering method, which also performs linear interpolation between mipmaps.
Texturing Functions

```c
void glTexEnv{if}{v}(GLenum target, GLenum pname, TYPE param);
```

- **target** must be GL_TEXTURE_ENV.
- If **pname** is GL_TEXTURE_ENV_MODE, **param** can be GL_DECAL, GL_REPLACE, GL_MODULATE, or GL_BLEND, to specify how texture values are combined with the color values of the fragment.
- If **pname** is GL_TEXTURE_ENV_COLOR, **param** is an array of four floating-point values representing R, G, B, and A components, used only if the GL_BLEND texture function has been specified as well.

### Components

<table>
<thead>
<tr>
<th>Components</th>
<th>Decal Mode</th>
<th>Modulate Mode</th>
<th>Blend Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>undefined</td>
<td>( C = L_tC_f ),</td>
<td>( C = (1-L_t)C_f + L_tC_c ),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( A = A_f )</td>
<td>( A = A_f )</td>
</tr>
<tr>
<td>2</td>
<td>undefined</td>
<td>( C = L_tC_f ),</td>
<td>( C = (1-L_t)C_f + L_tC_c ),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( A = A_tA_f )</td>
<td>( A = A_tA_f )</td>
</tr>
<tr>
<td>3</td>
<td>( C = C_t )</td>
<td>( C = C_tC_f ),</td>
<td>undefined</td>
</tr>
<tr>
<td></td>
<td>( A = A_f )</td>
<td>( A = A_f )</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( C = (1-A_t)C_f + A_tC_t ),</td>
<td>( C = C_tC_f ),</td>
<td>undefined</td>
</tr>
<tr>
<td></td>
<td>( A = A_f )</td>
<td>( A = A_tA_f )</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** t -- texture, f -- fragment, c -- the values assigned with GL_TEXTURE_ENV_COLOR
Texture Objects

- Generate texture objects or names: `glGenTextures()`
- Initially bind texture objects to texture data, including the image arrays and texture properties: `glBindTexture()`
- More often used objects stay in memory (prioritize the objects).
- Bind and rebind texture objects, making their data available for rendering texture models
- When we call `glBindTexture()` with a texture name, all subsequent `glTex*()` commands that specify the texture and its associated parameters are saved in the memory
Assigning Texture Coordinates

• Texture coordinates can comprise one, two, three, or four coordinates. They’re usually referred to as the \( s, t, r, \) and \( q \) coordinates.

• For one-dimensional textures, you use the \( s \) coordinate; for two-dimensional textures, you use \( s \) and \( t \).

• The \( q \), like \( w \), is typically given the value 1 and can be used to create homogeneous coordinates.

• The command to specify texture coordinates, `glTexCoord*()`, is similar to `glVertex*()`, `glColor*()`, and `glNormal*()`. Usually, texture-coordinate values range between 0 and 1; values can be assigned outside this range, however, with the results described in “Repeating and Clamping Textures.”
Computing Approximate Texture Coordinates

• Polygon aspect ratio 3/2 (w/h); texture aspect ratio 1. To avoid distorting the texture: use texture coordinates of (0,0), (1,0), (1,2/3), (0,2/3)

• For example, a can 4 units tall and 12 units around (aspect ratio 3/1). Textures must have aspect ratio of $2^n$ to 1. You can simply not use the top third of the texture: let’s approximate the can by 30 polygons. We can use the following texture coordinates:
  – $(0,0)$, $(1/30,0)$, $(1/30,1/3)$, $(0,1/3)$
  – $(1/30,0)$, $(2/30,0)$, $(2/30,1/3)$, $(1/30,1/3)$
  – …
  – $(29/30,0)$, $(1,0)$, $(1,1/3)$, $(29/30,1/3)$

• Only a few curved surfaces (cone and cylinders) can be mapped to a flat surface without geodesic distortion. For example, a sphere $(\cos \theta \cos \phi, \cos \theta \sin \phi, \sin \theta)$. The $\theta-\phi$ rectangle can be mapped directly to a rectangular texture map, but the closer you get to the poles, the more distorted the texture.
Repeating and Clamping Textures

• You can assign texture coordinates outside the range [0,1] and have them either clamp or repeat.

• With repeating textures, if you have a large plane with texture coordinates running from 0.0 to 10.0 in both directions, for example, you’ll get 100 copies of the texture tiled together on the screen.

• For most applications where the texture is to be repeated, the texels at the top of the texture should match those at the bottom, and similarly for the left and right edges.

• To clamp the texture coordinates: Any values greater than 1.0 are set to 1.0, and any values less than 0.0 are set to 0.0. Clamping is useful for applications where you want a single copy of the texture to appear on a large surface.

• If the surface-texture coordinates range from 0.0 to 10.0 in both directions, one copy of the texture appears in the lower corner of the surface. The rest of the surface is painted with the texture’s border colors as needed.
glBegin(GL_POLYGON);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0)
    glTexCoord2f(0.0, 3.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(3.0, 3.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(3.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
    glEnd();

glBegin(GL_POLYGON);
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 3.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(3.0, 3.0); glVertex3f(2.41421, 1.0, -1.41421);
    glTexCoord2f(3.0, 0.0); glVertex3f(2.41421, -1.0, -1.41421);
    glEnd();
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_S_WRAP, GL_REPEAT);

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_T_WRAP, GL_REPEAT);

or

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_S_WRAP, GL_CLAMP);

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_T_WRAP, GL_CLAMP);

You can also clamp in one direction and repeat in the other.
OpenGL texture mapping

1) Specify texture
   - read or generate image; gluScaleImage();
   - Assign to texture
   - glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0,
     GL_RGB, GL_UNSIGNED_BYTE, myImage);

2) Specify texture mapping parameters
   - Wrapping, filtering, how to combine pixel and texel, etc.

3) Enable GL texture mapping (GL_TEXTURE_2D)

4) Assign texture coordinates to vertices

5) Transform texture coordinates

5) Draw your objects

6) Disable GL texture mapping
Put it all together

... glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
    ...
    glEnable(GL_TEXTURE_2D);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB,
        GL_UNSIGNED_BYTE, mytexture);
    ...
    glMatrixMode(GL_TEXTURE);  //texture transformation//
    glMatrixMode(GL_MODELVIEW);
    ...
    DrawObjects ();  // define texture coordinates and vertices in the function
    ...
Automatic Texture-Coordinate Generation

- To make contours on a 3D model
- To simulate reflections from an arbitrary environment on a shiny model

```c
glTexGen*(GLenum coord, GLenum pname, TYPE param);
glTexGen*v(GLenum coord, GLenum pname, TYPE *param);
```

- GL_S or GL_T; coord to be auto. generated
- GL_TEXTURE_GEN_MODE
- GL_OBJECT_PLANE
- GL_EYE_PLANE
- GL_OBJECT_LINEAR, GL_EYE_LINEAR, or GL_SPHERE_MAP

If eye_plane, modelview matrix may not be the same when the polygon vertices are transformed. This function establishes a field of texture coordinates that can produce dynamic contour lines on moving objects.

Teapot

Control Keys

E, O, S, X

Copyright @ 2002 by Jim X. Chen:
jchen@cs.gmu.edu
Automatic Texture-Coordinate Generation

- If the texture generation function is `GL_OBJECT_LINEAR`, the following function is used to calculate for the parameter specified.
  - \( g = p_1x_0 + p_2y_0 + p_3z_0 + p_4w_0 \), where \((P_1, P_2, P_3, P_4)\) is the plane equation, and \((x_0, y_0, z_0, w_0)\) is the vertex for calculation. Here \( g \) stands for \( s \) or \( t \) in the texture coordinates.

\[
v = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \begin{pmatrix} x_0 & y_0 & z_0 \end{pmatrix} + \begin{pmatrix} x, y, z \end{pmatrix}
\]

\[
D = \frac{ax_0 + by_0 + cz_0 + d}{\sqrt{a^2 + b^2 + c^2}}
\]

Teapot

Control Keys

E, O, S, X
Automatic Texture-Coordinate Generation

• 1\textsuperscript{st}: specify the texture coordinate generation functions for S and T:
  – glTexGeni( GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR );
  – glTexGeni( GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR );

• 2\textsuperscript{nd}: enable texture coordinate generation:
  – glEnable( GL_TEXTURE_GEN_S );
  – glEnable( GL_TEXTURE_GEN_T );

• 3\textsuperscript{rd}: generate the texture coordinates. For object or eye linear, they are based on a reference plane:
  – static GLfloat sgenparams[] = { 2, 0, 0, 0 };
  – static GLfloat tgenparams[] = { 0, 1, 0, 0 };
  – glTexGenfv( GL_S, GL_OBJECT_PLANE, sgenparams );
  – glTexGenfv( GL_T, GL_OBJECT_PLANE, tgenparams );
Perspective correction

- Equal spacing in screen (pixel) space is **not** the same as in texture space in perspective projection
  - Perspective foreshortening
Perspective-Correct Texture Coordinate Interpolation

• Interpolate (tex_coord and \(w\)) over the polygon, then do perspective divide after interpolation

• Compute at each vertex after perspective transformation
  – “Numerators” \(s/w\), \(t/w\)
  – “Denominator” \(1/w\)

• Linearly interpolate \(1/w\), \(s/w\), and \(t/w\) across the polygon

• At each pixel
  – Perform perspective division of interpolated texture coordinates \((s/w, t/w)\) by interpolated \(1/w\) (i.e., numerator over denominator) to get \((s, t)\)
Perspective Correction Hint

- Texture coordinate and color interpolation:
  - Linearly in screen space (wrong) OR
  - Perspective correct interpolation (slower)

- `glHint (GL_PERSPECTIVE_CORRECTION_HINT, hint)`, where `hint` is one of:
  - GL_NICEST: Perspective
  - GL_FASTEST: Linear
  - GL_DONT_CARE: Linear