Chapter 15

Texture Mapping

Texture mapping assigns two additional coordinates s, t called *texture coordinates* at each vertex. These texture coordinates together indicate the horizontal and vertical position of a pixel in the *texture image* whose color should be used at that vertex position.

The texture coordinates s, t usually each range from zero to one. The texture image is a square array of pixels with integer $H \times V$ resolution. Hence the integer pixel referenced by texture coordinates (u, v) would be $(\text{round}(\frac{s}{H-1}), \text{round}(\frac{t}{V-1}))$.

As we showed in the last chapter, rasterization interpolates attribute values set at the vertices, spreading them evenly across the interior of the polygon. Texture mapping interpolates these texture coordinates s, t across the interior of the polygon, and each interior pixel uses as its color that of the pixel corresponding to the interpolated coordinates s, t.

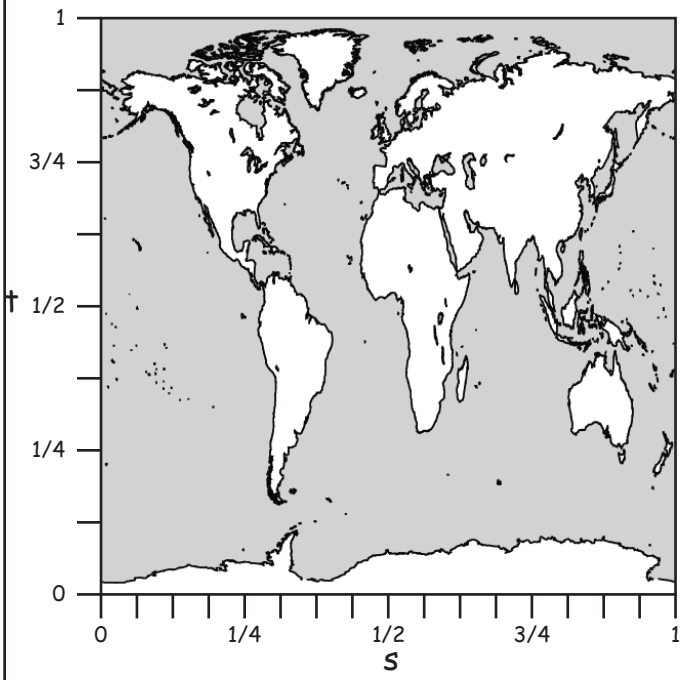
A familiar example of texture mapping is the process of mapping an atlas of the world onto a globe. An atlas uses a Mercator projection, which distorts the earth's spherical latitude and longitude coordinates to flat rectilinear coordinates, simply by setting $x =$ longitude and $y =$ latitude. Recall longitude ranges from $-180°$ to $180°$ and longitude ranges from $-90°$ to $90°$. We load the atlas into a square texture image, which will be referenced by texture coordinates u, v ranging from zero to one. Hence, for each vertex of the globe's polygonal mesh, we assign its s texture coordinate as a "longitude" ranging from zero to one, and its t texture coordinate as a "latitude" ranging from zero to one, specifically

$$u = (\text{longitude} + 180)/360, \qquad v = (\text{latitude} + 90)/180. \tag{15.1}$$

Each polygon of the globe mesh maps to a corresponding polygonal region in the atlas (texture image). The color of each pixel rasterized in the interior of this polygon region is drawn from the color of a position in the atlas (texture image) corresponding to the s, t texture coordinates of the polygon pixel interpolated from the texture coordinates stored at the polygon's vertices.

Texture Mapping

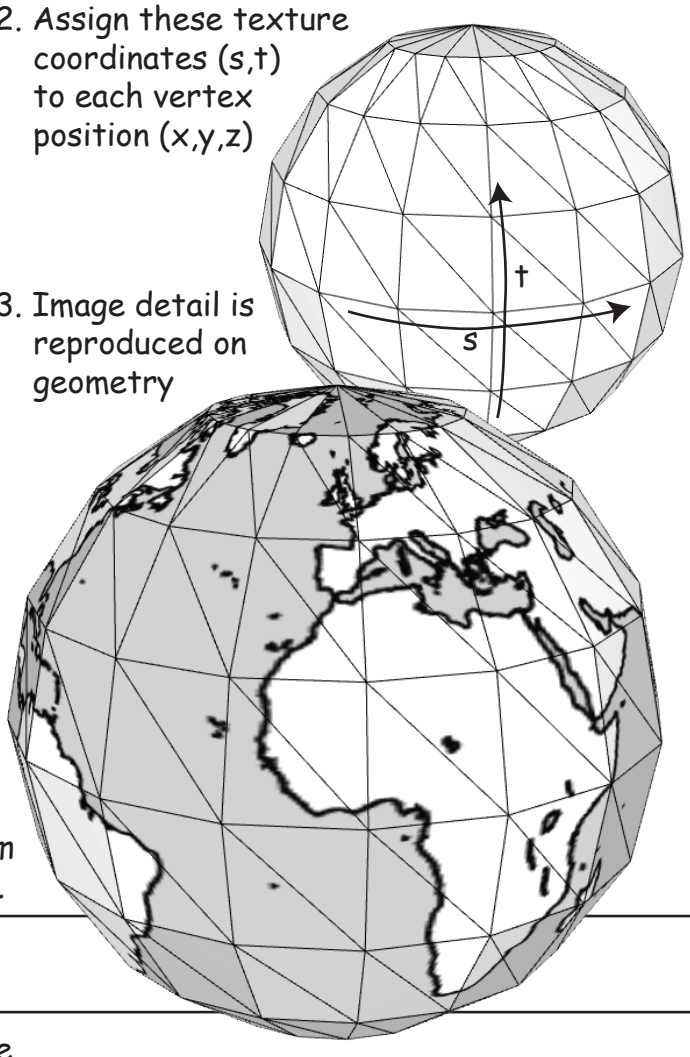
1. Texture coordinates (s,t) reference a position in a texture image



In this example, texture coordinates (s,t) are propotional to longitude,latitude but range from 0 to 1 instead of -180° to 180° and -90° to 90°.

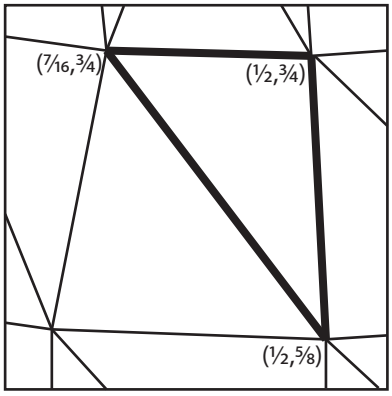
2. Assign these texture coordinates (s,t) to each vertex position (x,y,z)

3. Image detail is reproduced on geometry

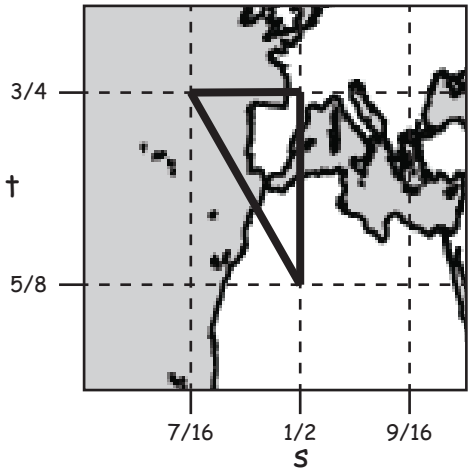


Each triangle's texture coordinates determine which part of the texture image to reproduce on the face of the triangle

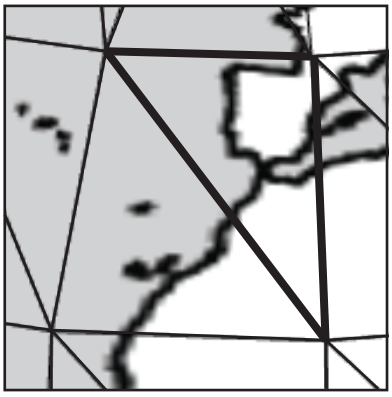
Geometry



Texture Image



Texture Mapped Geometry



Texture mapping works for more than just spherical shapes, so long as one can find an analog for latitude and longitude on the shape. Furthermore, the texture map need not cover the entire shape, and portions of the texture image can appear multiple times on a shape if the same texture coordinates are used on multiple regions of the mesh.

15.1 Texture Coordinates

One of the main challenges for texture mapping is the generation of the texture coordinates needed to map a two-dimensional texture image onto a surface embedded in three dimensions. Even if we assume the surface is a two-manifold (e.g. a two dimensional rubber sheet deformed to undulate through three dimensions) we still have the problem that the surface is specified only by the connectivity of 3-D vertices.

Some properties that a texture mapping might posess are worth noting. A texture mapping might be , which means every point on the surface corresponds to a unique point in the texture image. A texture mapping need not be one-to-one, and the same portion of the texture image can appear in multiple places on a texture mapped surface.

A texture mapping might also be , which means that every portion of the texture image for any s, t ranging over $[0, 1]^2$ appears somewhere on the texture mapped surface. A texture mapping need not be onto, and can use just a portion of the available texture image.

A texture mapping might also be , which means that neighboring points on the surface correspond to neighboring points in the texture image. A texture mapping need not be continuous. For example neighboring triangles can use assign different texture coordinates to a shared vertex, such that the texture coordinates interpolated across the two triangle faces do not need to agree at a shared vertex or edge. Such discontinuous vertices and edges form a texture mapping . A exture atlas is a discontinous texture mapping that takes a texture image consisting of multiple portions and maps it them different surface areas. EXAMPLE.

In ideal circumstances, the texture coordinates can be assigned when an object is modeled. Often an object's surface is defined by parametric functions and the function's parameters can be scaled or otherwise transformed to serve as texture coordinates. For example, (15.1) transformed the latitude and longitude parameters used to set the globe's spatial coordinates into unit ranges for use as texture coordinates.

When a model's vertices do not include texture coordinates, then they have to be assigned, either manually or automatically. There are several simple strategies for automatically generating two dimensional texture coordinates from three dimensional vertex positions. A common strategy is to set the texture coordinates s, t of a vertex to be a linear function of its

spatial coordinates (x, y, z),

$$s = A_s x + B_s y + C_s z + D_s, \quad (15.2)$$
$$t = A_t x + B_t y + C_t z + D_t. \quad (15.3)$$

While the A, B, C and D terms define plane equations, it is better to think of them as just parameters of linear functions. For example setting $(A_s, B_s, C_s, D_s) = (1, 0, 0, 0)$ and $(A_t, B_t, C_t, D_t) = (0, 1, 0, 0)$ defines the texture coordinates as $s = x$ and $t = y$.

Defining the texture coordinates as a linear function of spatial model coordinates has the effect of carving a shape out of an extrusion of the 2-D texture. For example, using $s = x$ and $t = y$ to define the texture coordinates for a texture image of concentric circles of woodgrain would result in the shape carved out of a block of wood whose grain direcion was oriented along the z axis. WOOD EXAMPLE.

These texture coordinate generation functions, such as this linear function, are commonly applied to spatial coordinates from the model coordinate system. However, they can also be applied to other coordinate systems, such as viewing coordinates to simulate some lighting, shading and contouring effects. CONTOURING EXAMPLE.

More sophisticated functions can be used to generate texture coordinates. For example, cylindrical texture coordinates can be generated as

$$s = (\text{atan2}(y, x) + \pi)/2\pi, \quad (15.4)$$
$$t = z, \quad (15.5)$$

where $\text{atan2}(y, x)$ returns the angle about the origin of the point (x, y) in radians ranging from $(-\pi, \pi]$. Any affine transformation can be applied to the spatial coordinates before conversion to cylindrical coordinates to adjust the orientation, placement and scale of the cylindrical coordinates. Similary, spherical coordinates can be generated by substituting

$$t = \text{acos}(z/\sqrt{x^2 + y^2 + z^2})/\pi, \quad (15.6)$$

where acos returns values in the range $[0, \pi)$.

15.2 Perspective Correct Texture Mapping

As we discussed in the previous section, texture coordinates (and other attribute values) are interpolated from their corner values at each vertex to their intermediate values at each fragment. When two vertices of a triangle lie at different distances from the image plane, the these values should not be simply linearly interpolated across the screen.

We can demonstrate why with the example of a sidewalk. Using texture mapping, we will model our sidewalk with a single quad with vertices in

viewing coordinates, and corresponding texture coordinates

$$\mathbf{v}_1 = (-1,-1,-1), \qquad s_1 = 0, \qquad t_1 = 0 \tag{15.7}$$
$$\mathbf{v}_2 = (1,-1,-1), \qquad s_2 = 1, \qquad t_2 = 0 \tag{15.8}$$
$$\mathbf{v}_3 = (1,-1,-10), \qquad s_3 = 1, \qquad t_3 = 1 \tag{15.9}$$
$$\mathbf{v}_4 = (-1,-1,-10), \qquad s_4 = 0, \qquad t_4 = 1. \tag{15.10}$$

We can then consider how to apply a texture image onto this sidewalk receding into the distance. The perspective projection of this viewing coordinate rectangle onto the viewing window is a trapezoid, and the vertices of this trapezoid are eventually assigned pixel locations in viewport coordinates. If we simply interpolate the texture coordinates using the viewport coordinates of the vertex positions, then the details of the texture vary regularly from the bottom to the top of the viewport trapezoid.

Recall that the perspective projection matrix converts a viewing coordinate point (x, y, z) to the window coordinate point $(x/w, y/w)$ where $w = -z/d$, since

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -z/d \end{bmatrix} \equiv \begin{bmatrix} \frac{x}{-z/d} \\ \frac{y}{-z/d} \\ -d \\ 1 \end{bmatrix}. \tag{15.11}$$

Let A be a vertex with viewing coordinates (x_A, y_A, z_A) and texture coordinates (s_A, t_A), which projects to the point $(x_A/w_A, y_A/w_A)$ where $w_A = -z_A/d$. Similarly let B be a vertex with viewing coordinates (x_B, y_B, z_B) and texture coordinates (s_B, t_B), which projects to the point $(x_B/w_B, y_B/w_B)$ where $w_B = -z_B/d$. Let the point P on the line between A and B in viewing coordinates, such that $P = (1-\alpha)A + \alpha B$ for some α between zero and one.

The point P projects to $(x_P/w_P, y_P/w_P)$. The coordinates $x_P = (1-\alpha)x_A + \alpha x_B$ and similary for y_P. Likewise, $w_P = (1-\alpha)w_A + \alpha w_B$ since $z_P = (1-\alpha)z_A + \alpha z_B$ and d is constant. Thus for each value α we can find x_P, y_P and w_P ...

Let A be a pixel position in viewport coordinates corresponding to vertex $\mathbf{v}_A$ in viewing coordinates, such that $A = WP\mathbf{v}_A$, and similarly for B. Let P be a viewport coordinate position along a straight (non-rasterized) line from A to B such that $P = (1-\alpha)A + \alpha B$ for some α between zero and one. Note that $A = W[x_A/w_A, y_A/w_A, d, 1]$ where $w_A = -z_A/d$, and similar for B. Then

$$\begin{aligned} P &= (1-\alpha)A + \alpha B && (15.12)\\ &= (1-\alpha)W[x_A/w_A, y_A/w_A, d1] + \alpha W[x_B/w_B, y_B/w_B, d1] && (15.13)\\ &= W\left((1-\alpha)[x_A/w_A, y_A/w_A, d1] + \alpha[x_B/w_B, y_B/w_B, d1]\right) && (15.14)\\ &= W\left((1-\alpha)[x_A/w_A, y_A/w_A, d1] + \alpha[x_B/w_B, y_B/w_B, d1]\right) && (15.15)\\ & && (15.16) \end{aligned}$$

15.3 Texture Sampling

As we have already seen in Chapter **??** on Rasterization, often computer graphics processes result in aliasing. The term "alias" is colloquially used to describe an alternative name that disguises one's true identity. Aliasing means precisely the same thing in signal processing (and thus computer graphics) when inadequate sampling makes a signal appear differently than it truly is.

SINE WAVE EXAMPLE

A classic example of aliasing results from digitizing a sine wave poorly. We digitize a signal by measuring it at a finite number of discrete points (and quantizing the resulting measurement). These discrete measurements are called samples, and often a signal is sampled at regular intervals. These samples can then be used to reconstruct the signal, by connecting the dots to estimate the signal from which they were drawn.

If a signal is sampled by too few regular measurements, then the resulting signal reconstructed from those samples can appear quite different. Often the kind of signal is unchanged (e.g. a sine wave) but its behavior or parameters (e.g. its frequency) can be quite different. The reconstructed signal is an alias, disguising the true identity of the original signal.

Shannon's sampling theorem states that the sampling rate should be at least twice the signal's highest frequency (it's so-called Nyquist limit), but this form requires significanly more signal processing theory to properly define and prove. A gross simplification that will work for our purposes here is that a signal should be sampled at least twice as many times as it wiggles.

Aliases often arise when digitizing an analog signal, but they also arise when redigitizing a digitized signal, often called resampling. Texture mapping is such a resampling process, as we are using the colors of a digitized image in the rendering of another digitized image. Aliases arise because the resolutions of these digitized images rarely agree, as shown in Figure 15.1.

We will study the reconstruction of a texture image mapped onto a displayed surface with texture coordinates s, t as a sampling of one or more texture image pixels corresponding to these coordinates. We will assume our texture is square with resolution $n \times n$ (where n is usually a power of two). More formally, we will define this reconstruction by defining a sampling function sample(s, t) assigned texture coordinates s, t, (real values ranging from zero to one) as the combination of one or more texture samples texture(i, j) where i and j are integers ranging from zero to $n - 1$. We will also use the real values

$$S = s \times (n - 1) \quad (15.17)$$

$$T = t \times (n - 1) \quad (15.18)$$

as a shorthand to indicate the real position amid the texture image pixel

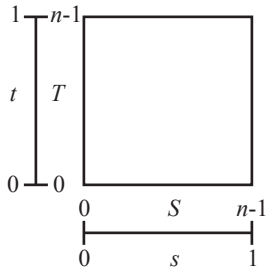


Figure 15.2: The ranges of texture coordinates s, t and their corresponding ranges of texture image pixel locations S, T.

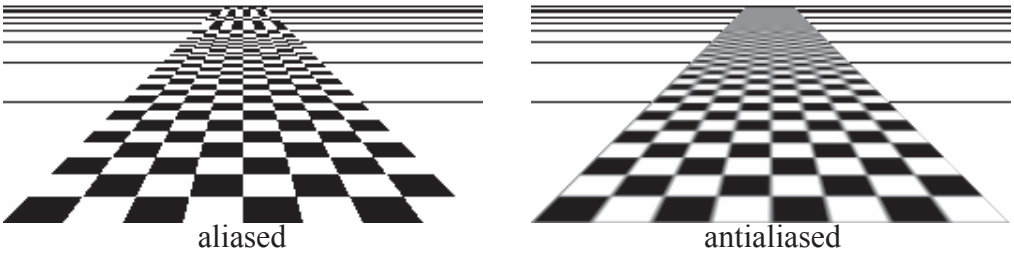


Figure 15.1: Aliased (left) v. antialiased (right) rendering of a (1x9) quad receding in perspective, textured with a checkerboard pattern. The lines on its sides mark distances from the viewer in one unit increments. Aliases at the near end occur as blocky staircasing whereas aliases at the far end make the checkerboard pattern appear as a different pattern (e.g. striped).

location corresponding to the texture coordinates.

We will examine two main cases of mismatch between the texture resolution and the display resolution. Magnification aliasing occurs when texture is magnified, a lower resolution texture is mapped onto a higher resolution display. Minification aliasing occurs when a texture is reduced, a higher resolution texture is mapped onto a lower resolution display. Both exhibit aliasing artifacts that we can reduce, beginning by properly understanding the problem.

Magnification Aliasing

Texture magnification occurs when we have too few texture pixels, which are mapped to many more display pixels. This magnification blows up texture pixels and displays each one using several display pixels. Texture magnification exhibits aliases not because we are using too few display pixels to sample the texture, but because we are using too few texture samples to represent some underlying signal and the resulting blockiness of the texture pixels is magnified when mapped to the display pixels.

Nearest Neighbor Reconstruction

Nearest neighbor reconstruction sets the color of a display pixel to the texture pixel that maps closest to it. This is easily implemented by rounding a display pixel's texture coordinates to its nearest discrete value.

The nearest neighbor of a real (e.g. floating point) number is just its nearest integer value, rounding up for values halfway between integer values

$$\text{round}(x) = \lfloor x + \frac{1}{2} \rfloor. \tag{15.19}$$

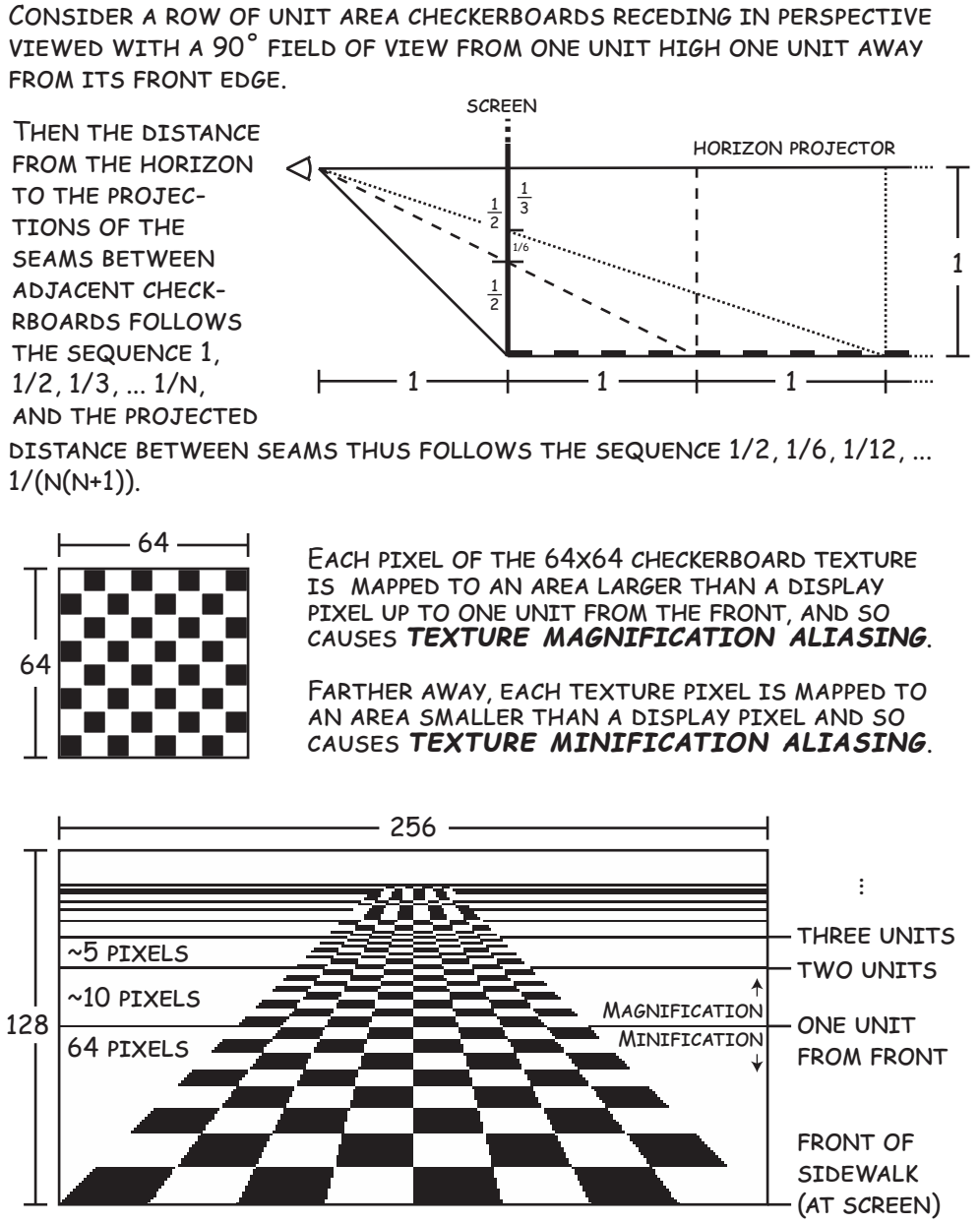


Figure 15.3: Texture minification and magnification aliasing demonstrated on a checkerboard texture.

For a texture coordinate, say the horizontal texture coordinate s, then we want to map the real domain of s, from zero to one, to the integer range of n texture map pixel locations, from zero to $n-1$. Hence the location of the nearest neighbor sample for one texture coordinate is

$$\mathrm{nn}(s) = \mathrm{round}(S) = \mathrm{round}((n-1)s). \tag{15.20}$$

Figure 15.4 demonstrates this mapping for texture that is eight pixels wide.

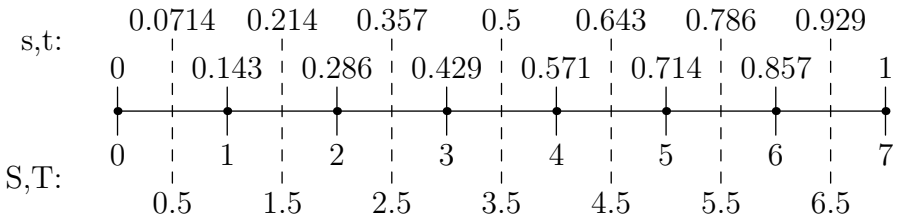


Figure 15.4: Mapping between a texture coordinate s and a texture whose width is eight pixels, along with the boundaries defined by nearest neighbor reconstruction.

If rasterization assigns the current display pixel with texture coordinates s, t, then nearest neighbor reconstruction would assign its color as

$$\text{sample}_{\text{NN}}(s,t) = \text{texture}(\text{nn}(s), \text{nn}(t)). \tag{15.21}$$

For example, for a 4×4 texture image, $\text{sample}_{\text{NN}}(3/4, 7/8) = \text{texture}(2, 3)$ since texture coordinate $s = 3/4$ rounds down to integer texture image coordinate 2 whereas texture coordinate $t = 7/8$ rounds up to integer texture image coordinate 3.

Nearest neigbor reconstruction results in a blocky appearance. This is because the domain of texture coordinates s, t that return the same texture image pixel is square. Hence each texture pixel will appear as a (possibly deformed) block spanning several pixels on the display (depending on the distortion induced by the assignment of texture coordinates to the displayed geometry.)

Bilinear Reconstruction

Since almost all real texture coordinates s, t lie between the finite number of texture coordinates that map precisely to integral texture image pixel positions, it makes sense to mix the colors of several nearby texture image pixels instead of just using the color of the closest one. Bilinear reconstruction combines four nearby texture image pixel values, weighted such that closer texture image pixels influence the color more than do more distant ones.

We measure "distance" along each coordinate separately. Let

$$\text{frac}(x) = x - \lfloor x \rfloor \tag{15.22}$$

return the fractional part of a real value x.

We abreviate

$$fs = \text{frac}(s(n-1)), \tag{15.23}$$

$$ft = \text{frac}(t(n-1)). \tag{15.24}$$

Then fs is the distance from the nearest smaller real s coordinate that maps to an integer texture image coordinate, and $1 - fs$ is the distance

from the nearest larger real s coorinate that maps to an integer texture image coordinate. Likewise for ft and $1-ft$.

Bilinear reconstruction is thus implemented as

$$\begin{aligned}\text{sample}_{\text{BL}}(s,t) =&(1-\text{frac}(S))(1-\text{frac}(T))\text{texture}(\lfloor S\rfloor,\lfloor T\rfloor)+\\ &\text{frac}(S)(1-\text{frac}(T))\text{texture}(\lfloor S\rfloor+1,\lfloor T\rfloor)+\\ &(1-\text{frac}(S))\text{frac}(T)\text{texture}(\lfloor S\rfloor,\lfloor T\rfloor+1)+\\ &\text{frac}(S)\text{frac}(T)\text{texture}(\lfloor S\rfloor+1,\lfloor T\rfloor+1).\end{aligned} \tag{15.25}$$

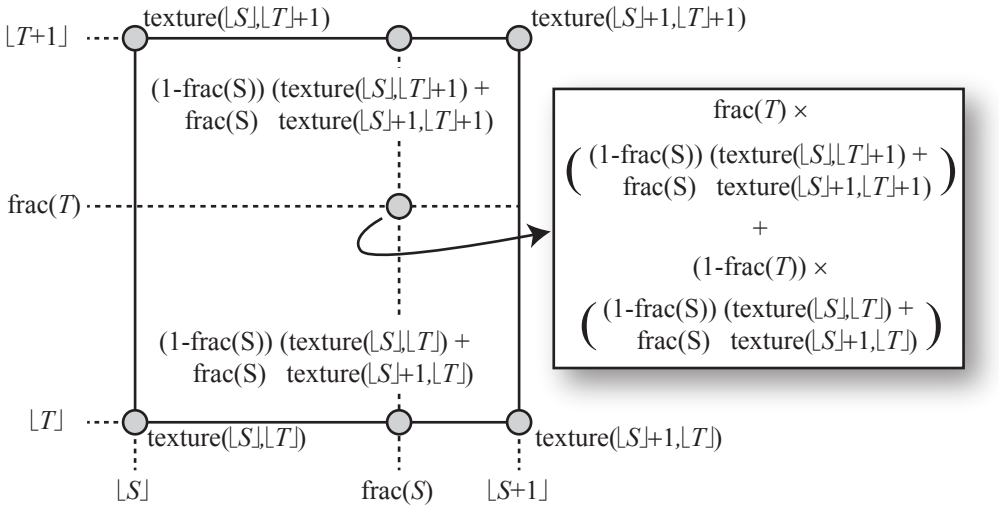


Figure 15.5: Bilinear filtering determines the color of a texture sample located between texture image pixel locations by weighting the colors of texture image pixels.

Figure 15.6 compares nearest neighbor and bilinear interpolation texture magnification filters.

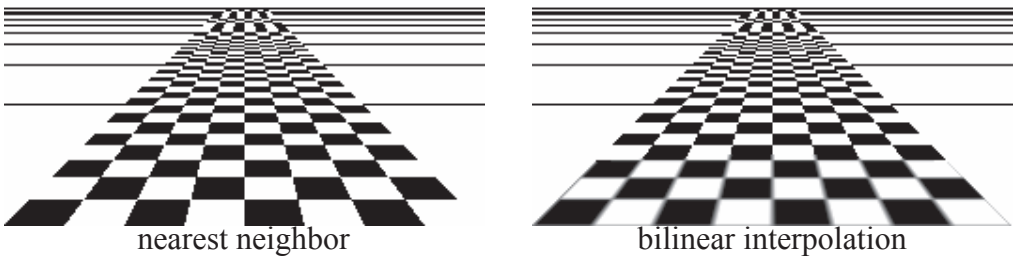


Figure 15.6: Comparison of texture magnification filters. Nearest neighbor (left) exhibits blocky stairstep artifacts whereas bilinear filtering blurs the boundary between checkerboard squares.

Minification Aliasing

Texture minification happens when we have more than enough texture pixels that map to too few display pixels. Texture minification results in aliasing because we are sampling the texture image with too few image pixels.

Since each display pixel needs to represent the combined color of several texture pixels, we want to display the average color of the region of texture image pixels that map to the domain of the a display pixel. However, we cannot compute this average on the fly, not only because it remains too costly, but mainly because it would require an arbitrary and varying amount of computation for each display pixel, which reduces SIMD compute efficiency and delays rendering. Instead we strive to precompute estimates of these averages that can be accessed in constant time.

The MIP map provides an estimate of an average texture pixel value across a region of pixels. Given an $n \times n$ square texture image (recall $n = 2^m$ for some positive m), we will label this texture_0. We then create a "pyramid" of textures, each half the resolution (horizontally and vertically) of the previous, labeled texture_k as k increases from 0 to m. Each pixel of texture_k represents the average color of four pixels of texture_{k-1}, specifically as

$$\text{texture}_k(i,j) = \frac{1}{4}\begin{pmatrix} \text{texture}_{k-1}(2i,2j)+ \\ \text{texture}_{k-1}(2i+1,2j)+ \\ \text{texture}_{k-1}(2i,2j+1)+ \\ \text{texture}_{k-1}(2i+1,2j+1) \end{pmatrix}. \tag{15.26}$$

This process concludes with texture_m, which consists of a single pixel whose color represents the average of all of the pixels in the original texture image. Hence MIP is an acronym for the latin phrase *multim in parvo*, which means "many things in a small place" [**?**].

Note that we have indexed the MIP maps such that the resolution of texture_k is $2^{(m-k)}$ and each pixel of texture_k represents a square of $2^k \times 2^k$ pixels in the original texture image.

MIP maps reduce minification aliases by picking the texture image resolution that most appropriately matches the display resolution. Recall that polygon rasterization interpolates texture coordinates with partial derivatives (e.g. $\partial s/\partial x$) that indicates the change in texture coordinate relative to the change in viewport coordinate. We can then set, for the current pixel, the real value

$$k = 1 + \lg \max\left(\frac{\partial s}{\partial x}, \frac{\partial s}{\partial y}, \frac{\partial t}{\partial x}, \frac{\partial t}{\partial y}\right) \tag{15.27}$$

The maximum of the partial derivatives estimates the number of texture image pixels that can map (in one direction) across the current display

pixel. If we set the MIP map level to the base-two log of this value, we get approximately one display image pixel per texture image pixel, but the Nyquist limit indicates that the texture resolution should be half the display resolution (horizontally and vertically). To accomodate this, we increment this MIP map level by one (which decreases the effective texture image resolution to half that of the display resolution) and assign it to k.

We have assigned to k a real value that we need to resolve to integer MIP map levels. As before, we can use nearest neighbor or linear interpolation to find one or more MIP map entries to evaluate our sample. We implement a "safest" neighbor MIP mapping as

$$\text{sample}_{SNMM}(s,t) = \text{sample}(s,t,\text{texture}_{\lceil k \rceil}) \tag{15.28}$$

where the righthand side indicates either nearest neighbor or bilinear interpolation of the texture sample from an integer MIP map level no less than k. We use this "safest" neighbor instead of simple rounding since we want to prevent aliasing by ensuring the texture resolution not exceed half the displayed resolution.

Similarly, we can implement linear interpolation of MIP map samples as

$$\begin{aligned}\text{sample}_{LMM}(s,t) = (1-\text{frac}(k))\ \text{sample}(s,t,\text{texture}_{\lfloor k \rfloor}) + \\ \text{frac}(k)\ \text{sample}(s,t,\text{texture}_{\lfloor k+1 \rfloor}).\end{aligned} \tag{15.29}$$

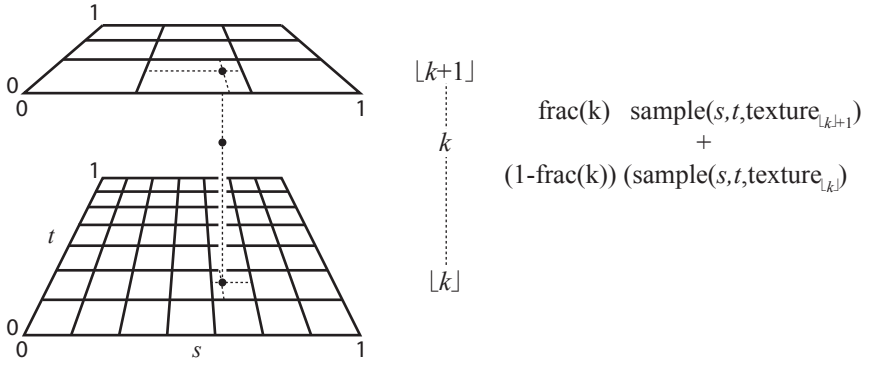


Figure 15.7: Trilinear interpolation of mip-mapped texture values. Texture samples are bilinearly interpolated at mip-map levels $\lfloor k \rfloor$ and $\lfloor k \rfloor + 1$ and then interpolated based on the fractional portion of k.

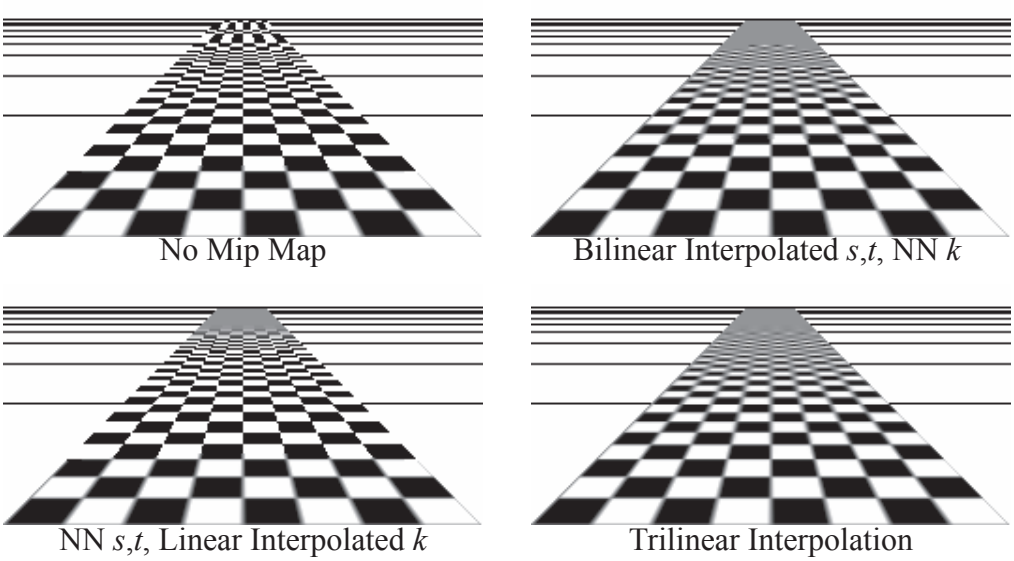


Figure 15.8: Combinations of filtering mip-map levels for reducing minification aliasing when the texture resolution exceeds its displayed resolution. When mip mapping is not employed (upper left), minification aliasing can be observed farther than one unit away from the front of the checkerboard, even though bilinear interpolation is used. When a single mip map level is chosen based on the number of pixels spanned by the projection of a texture image pixel (upper right), bilinear interpolation of neighboring texture image pixels at that level remove minification aliases but also over-blur the rendering at four units beyond the front of the checkerboard. Linear interpolation of nearest neighbor samples taken from the nearest coarser and finer mip map level (lower left) smoothly blends the transition from checkerboard to gray. Trilinear interpolation reduces minification aliases while preserving the checkboard pattern as far as four units beyond the front of the checkerboard.