Texturing

Andreas H. König

March 13, 2000

Enhancing the visual appearance of plain objects by applying definitions of fine structures to surfaces is called *Texturing*. Different material properties may be simulated:

- Color
- Reflection
- Gloss
- Transparency
- Bumps

The application of textures yields a more natural and realistic appearance of objects. It also enhances the 3D-impression of objects. The *texture gradient* defines the amount of deformation of the texture pattern due to the spatial position in 3D-space. Three properties may be controlled:

- Size
- Shape
- Density

Figure 1 shows examples for these variations.

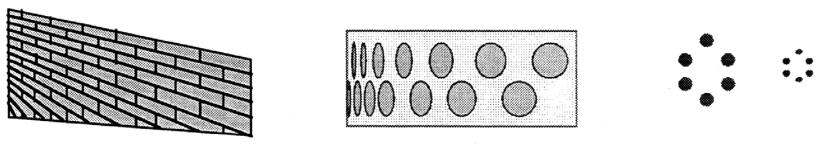

Figure 1: Texture deformations: Size, shape, and density.

The procedure of applying a texture to an object is called *mapping*. Two projection steps are used to define the mapping: *texture coordinates* (u, v) are projected into *object space coordinates* (x_0, y_0, z_0), which are defined on the surface of the object. Finally object space coordinates are mapped to *image space coordinates* (x, y). Figure 2 depicts the procedure. Textures usually are of rectangular shape in texture space. After the mapping transformation, they have been deformed to non-planar patches, bounded by four curves. Figure 3

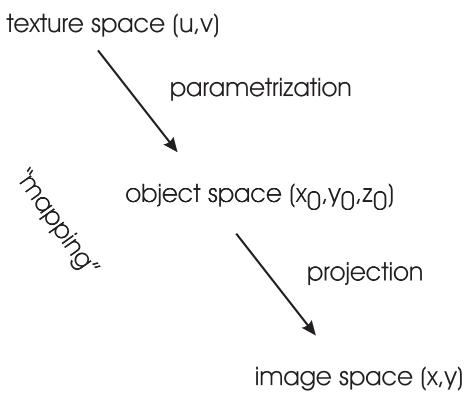

Figure 2: Texture mapping

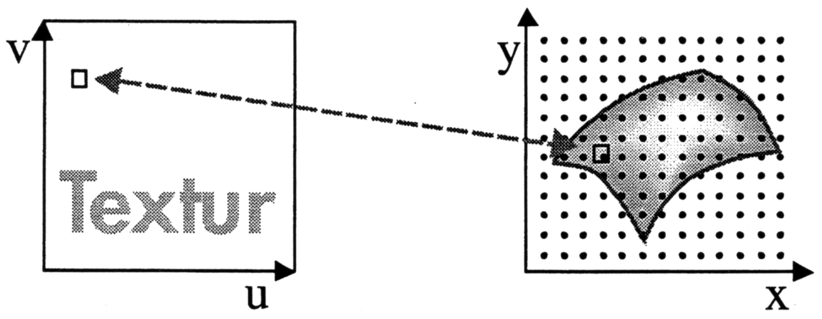

Figure 3: Texture space vs. image space

shows an example. The mapping procedure is usually defined in the opposite direction: for each pixel on the image plane, the color values of the according texture position have to be derived. Let $(x, y) = O(u, v)$ be the mapping definition for the object surface to the image space (respectively $x = O_x(u, v)$ and $y = O_y(u, v)$). The shading influence in image space for position (x, y) is defined by $S(x, y) = T(u, v)$, using the texture value of position (u, v) in texture space. Therefore the inverse mapping $S(x, y) = T(O^{-1}(x, y))$ has to be found.

1 Parametrization

The parametrizaton O defines the projection of the texture onto the object. A proper definition is vital (refer to Figure 4). The definition of the parametrization is discussed using the example of a simple triangle. Mapping parameters for each vertex of the triangle are defined in terms of texture coordinates (u, v). All points within the area of the triangle can be described by baricentric coordinates: $(x, y, z) = a_0 * (x_0, y_0, z_0) + a_1 * (x_1, y_1, z_1) + a_2 * (x_2, y_2, z_2)$. (a_0, a_1, a_2) are the baricentric coordiantes. The according texture coordinates (u, v) are

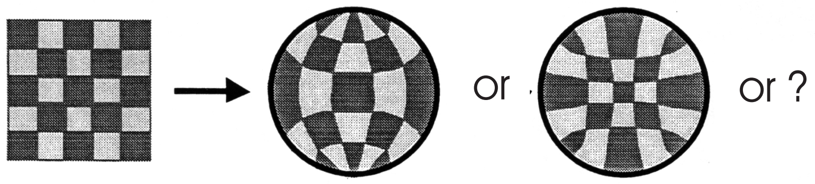

Figure 4: Definition of parametrization

easily derived by $(u, v) = a_0 * (u_0, v_0) + a_1 * (u_1, v_1) + a_2 * (u_2, v_2)$ (also refer to Figure 5). Arbitrary polygons can be divided into simple triangles for mapping.

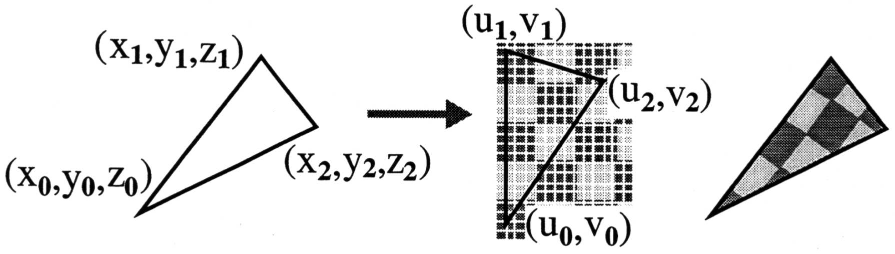

Figure 5: Parametrization for a triangle

A mapping for patch-surfaces is realized in a straight forward way by utilizing the patch parameters for texture coordinates (u, v) (refer to Figure 6).

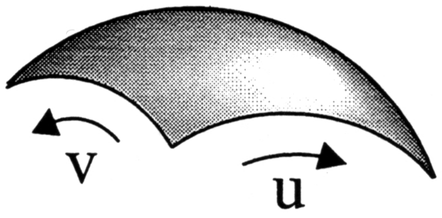

Figure 6: Parametrization for a patch

2 Types of textures

The most commonly used way of specifying a texture is by the definition of a regular 2D grid storing texture information just like an image bitmap. In analogy to the *pixels* of bitmaps, the term *texels* is used for the elements of such a texture.

Nevertheless, general texture definitions may be one-, two-, or three-dimensional.

3 Scanline oriented mapping

scanline oriented rendering methods are calculating images line by line. With the usage of textures, the inverse mapping function $O^{-1}(scanline)$ is needed. If O^{-1} is not available, the texture itself has to be deal with in a line by line way. This technique is refered to as *texture scanning*. I may result in holes in the calculated image or multiple evaluations of the same image regions. A better approach is *2-pass Scanning*. The mapping transformation is split into two one-dimensional shear mappings, which makes this method only suitable for affine and perspective transformations.

4 Solid texturing

Three-dimensional textures are usually defined as a function $T(x, y, z)$ in object space. This approach eliminated the parametrization step, yielding a undeformed texture. Nevertheless it is possible to specify the texture in 3D-parameter space like $T(u, v, w)$.

Solid objects with inner structure of the material (like wood or marble) can be easily simulated with solid textures. Here is an example for a wood texture definition:

$$\begin{array}{lcl} 0 \le x^2 + y^2 \le 1, zarbitrary & \rightarrow & lightbrown \\ 1 \le x^2 + y^2 \le 2, zarbitrary & \rightarrow & darkbrown \\ 2 \le x^2 + y^2 \le 3, zarbitrary & \rightarrow & yellow \\ etc. & & \end{array}$$

Figure 7 shows the influence of this texture to an object.

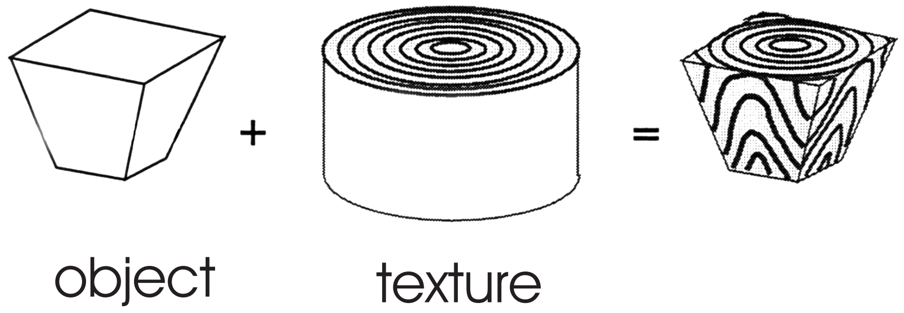

Figure 7: Pyramid coe with solid wood texture

Another approach within solid texturing is called *bombing*. A number of spheres with random size and location is defining a appearance of the texture similar to swiss cheese. Figure 8 shows an example.

5 Environment mapping

With this approach, the texture is defined an surrounding environment, which is shrink-wrapped to the object. In a preprocessing step, an environment map

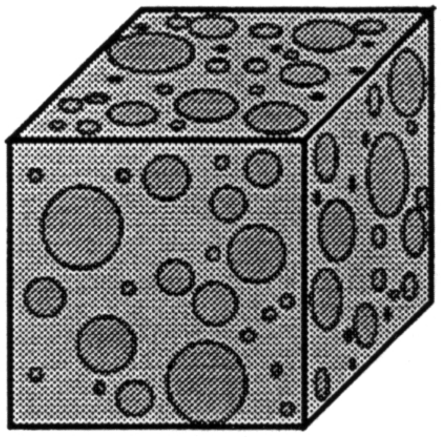
Figure 8: Bombing

is defined. The desired texture is projected to an surrounding hull of the object. As the surrounding sphere is much larger as the object to be mapped to, it is

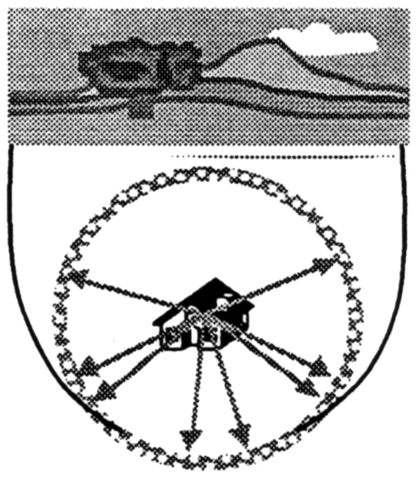
Figure 9: Environment mapping

sufficient to trace viewing rays from the center of the object to the sphere in order to determine texture entries. If the texture is defined in polar coordinates, the according texels can be accessed conveniently. For reflective surfaces, the intersection point of the reflected rays with the environment sphere can be used to determine the texture information. For diffuse surfaces the environment texture map has to be lowpass filtered beforehand, in order to account for the scattered light transport of diffuse reflection.

The environment may be shaped like a cube, sphere, or cylinder. The projection step for generating the environment map from a surrounding scene is different for each type. For the cube the map is generated with a rendering method taking perspective distortion in to account. Raytracing can be used to generate the map for the sphere, where just viewing rays are considered, but no reflected rays. Figure 10 shows the difference.

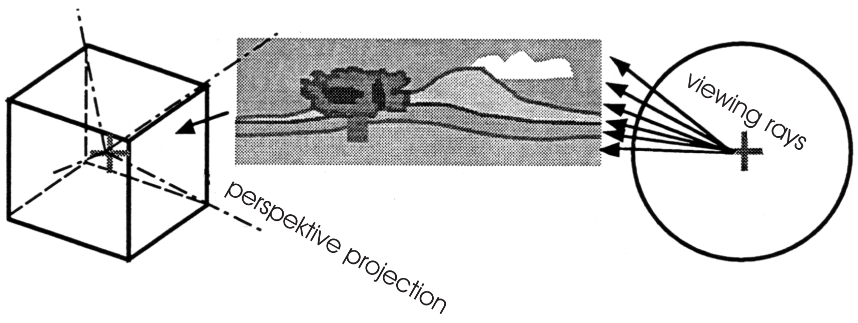

Figure 10: Environment map generation for cube and cylinder maps

6 Bump mapping

Bump mapping is able to simulate minor imperfections of surfaces. As modeling would be to resource consuming, the bumps are just simulated by shading the surface a little differently. Figure 11 shows the principle. As the lightness of

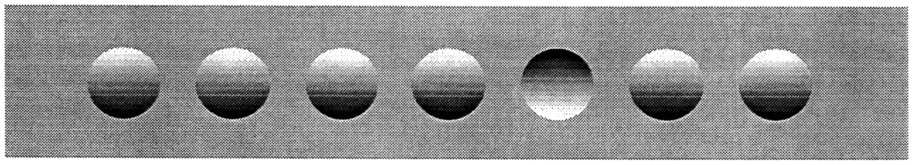

Figure 11: Bump mapping

a certain point is depending on the surface normal vector at this spot, bump mapping manipulates the normal vector. Texture is usually specified as a height field, where the new normals can be derived easily. Figure 12 shows the mapping procedure.

Please note, that bump mapping does not change the geometry of objects, but only the way, they are shaded. This means, that the outline of the objects still are rendered smoothly, as they are.

6.1 Horizon mapping

Horizon mapping tries to simulate real bump in a more sophisticated way than standard Bump mapping. For each texel eight horizon values are calculated, which are interpolated before shading. The point of consideration is only in

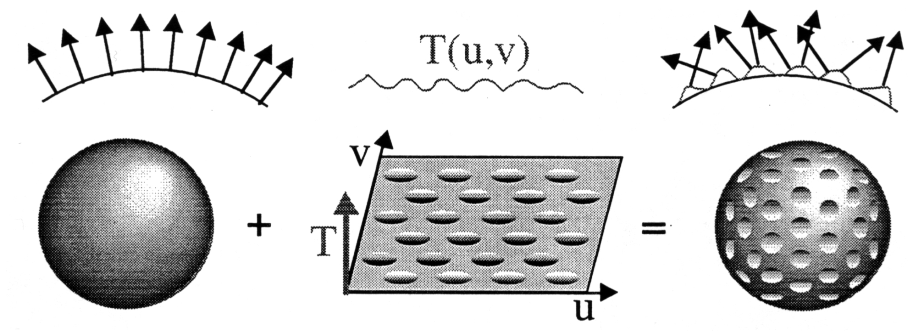

Figure 12: Using height fields for bump mapping

direct light, if the direction to the light source is higher than the pre-computed horizon. Figure 13 shows a sketch of the application of this technique, figure 14 shows an object rendered with this technique.

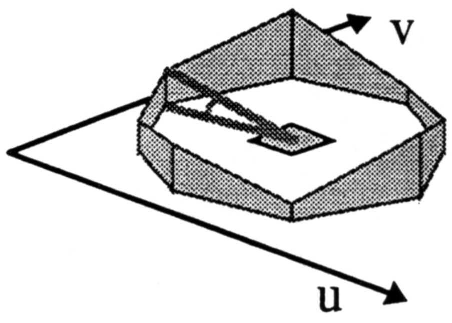

Figure 13: Horizon mapping

7 Texture anti-aliasing

Whenever textures are applied to objects, aliasing artifacts appear. Due to the deformation in terms of size, Moire-patterns or pixelization artifacts will show up. Figure 15 shows one of the problematic regions when projecting to spheres.

The reason for the effects of aliasing is to be found in the mapping process. If the projection of a image-space pixel covers more than one texel in texture space, it is not an appropriate strategy to chose just one of the covered texels for color evaluation. A better way to do it would be to user a weighted average of the covered texels. Figure 16 depicts the mapping situation.

This averaging can either be done while texture evaluation (*direct convolution*) or in a preprocessing step (*prefiltering*).

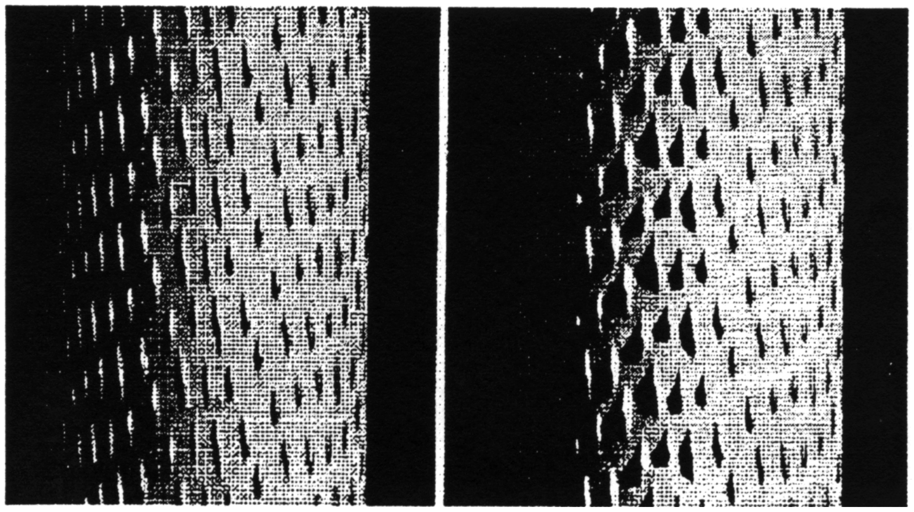

Figure 14: Two cylinders rendered with horizon mapping (right hand side) and without (left hand side)

7.1 Direct convolution

Due to the limited time constrains while rendering, just approximations of the projected pixels are used. Simple shapes like rectangles of ellipsoids are evaluated for generating the average of covered texels. Figure 17 gives examples.

This approach may be computationally expensive, yet it yields nice texture qualities.

7.2 Mip-Mapping

A more efficient method is the prefiltering approach *Mip-Mapping.* In a preprocessing step different resolutions of the texture are precalculated, each reduced version being half the size. The coarsest representation features a single texel. Figure 18 shows the reduction process.

The prefiltered textures can be efficiently stored using an scheme, which separates the RGB channels, shown in Figure 19.

In order to map the prefiltered texture to an object, the level of the Mip-Map D to be used has to be determined first. The maximum of the projected diagonals is used to calculate this level: $D = ld(max(|d_1|, |d_2|))$. Usually D will not be a natural number, therefore it has to be interpolated between the according smaller and larger maps:

$$colorofpixel = (D_1 - D) * W_0 + (D - D_0) * W_1 = (D_1 - D) * (W_0 - W_1) + W1$$

where W_0 is the value from map $D_0 = trunc(D)$ and W_1 from map $D_1 = D_0 + 1$, derived by linear interpolation (rfer to Figure 20.

If the original resolution is too small ($D < 0$), the texture has to be enlarged (*upsampling*) by interpolating the finest resolution map. Better quality cannot

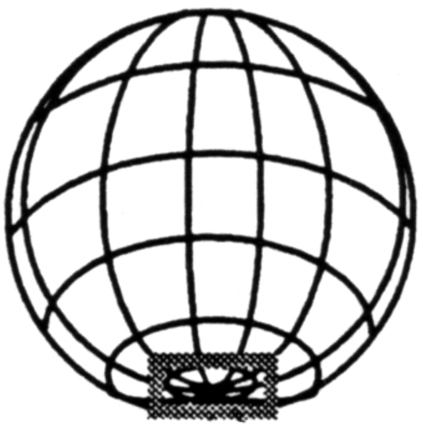

Figure 15: Very fine structures at the pole of the sphere will yield artifacts

be achieved, as the texture is not defined more accurately (refer to Figure 21):

$$\begin{aligned} T(u+du, v+dv) &= du * dv * T(u+1, v+1) + du * (1-dv) * T(u+1, v) + \\ & \quad (1-du) * dv * T(u, v+1) + (1-du) * (1-dv) * T(u, v) \end{aligned}$$

7.3 Summed area table method

Another prefiltering method is precaculating and storing the sum of all texels in the square $(0,0) - (u_0, v_0)$ (refer to Figure 22):

$$S(u_0, v_0) = \sum u \leq u_0, v \leq v_0 T(u, v)$$

If the sum of a certain rectangular region in texture space in needed, it can be derived from the precalculated sums with constant effort:

$$sum = S(u^+, v^+) - S(u^-, v^+) + S(u^-, v^-) - S(u^+, v^-)$$

Just for memory accesses and additions are necessary to compute the sum. Figure 23 shows an example.

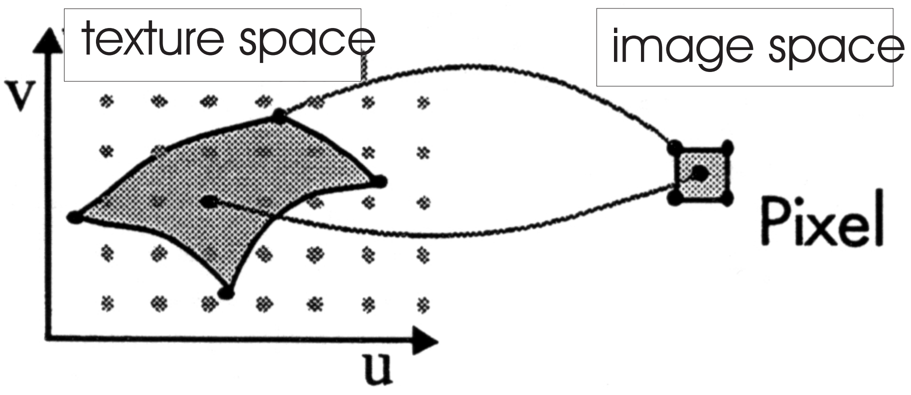

Figure 16: The origin of artifacts is to be found in the mapping procedure. If more than one texel is covered by the projected pixel, a weighted average of the texels should be used.

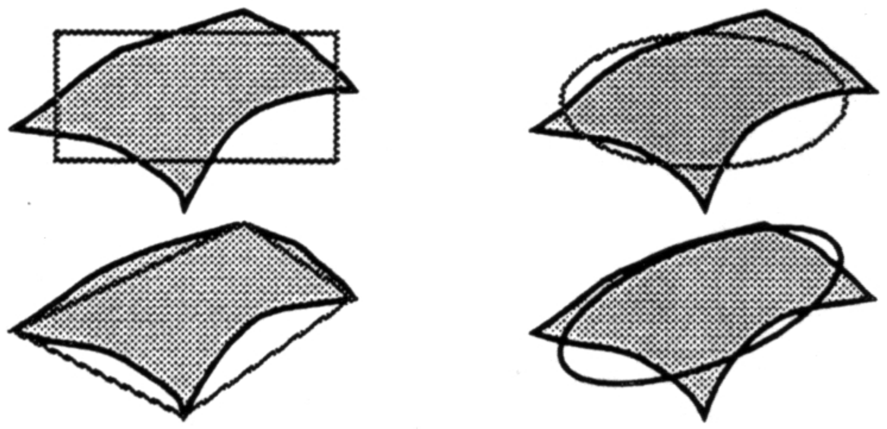

Figure 17: Direct convolution: simple shapes are used to approximate averaging of the covered area.

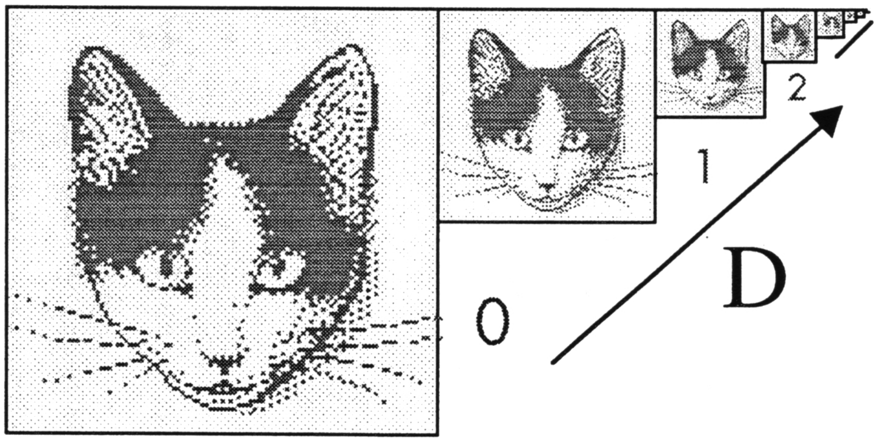

Figure 18: Mip-Mapping texture downsampling

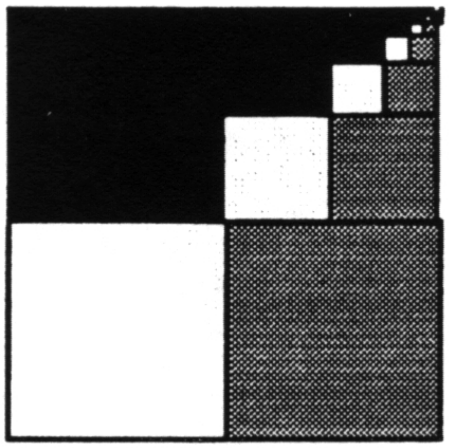

Figure 19: Mip-Mapping texture storage scheme

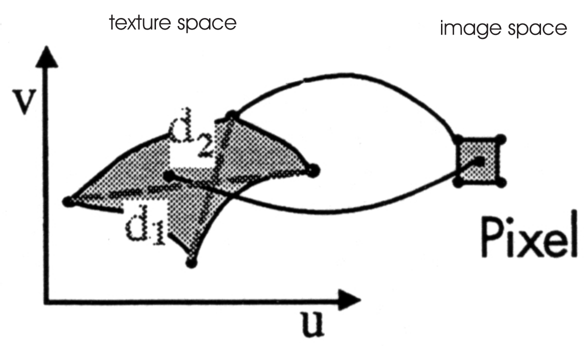

Figure 20: Mip-Mapping texture level derivation by length of diagonals

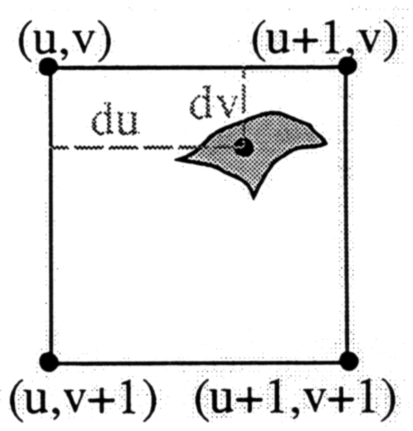

Figure 21: Mip-Mapping texture enlargement interpolation

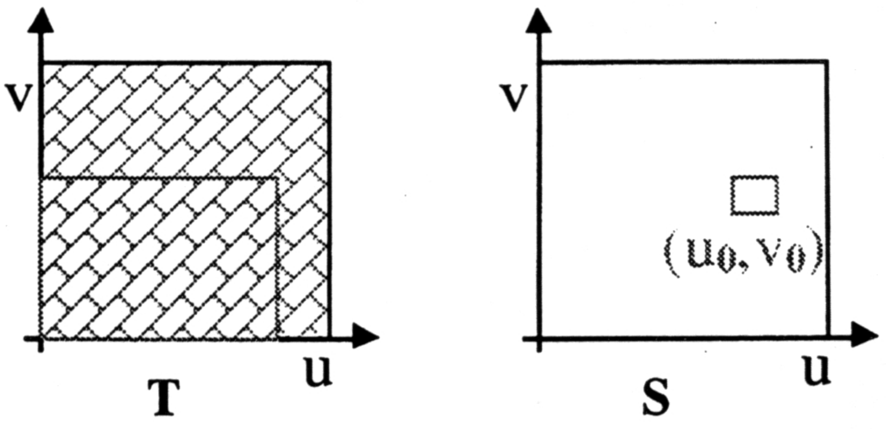

Figure 22: Summed area table generation

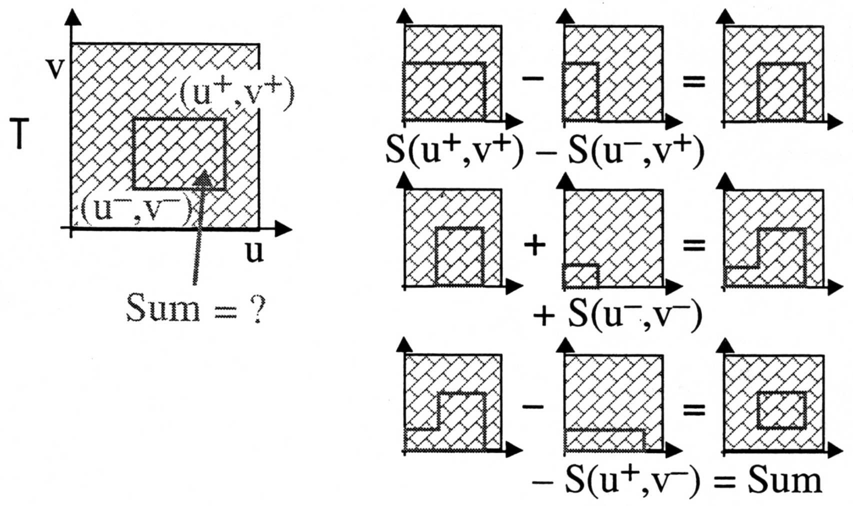

Figure 23: Summed area table access