Texturing

Peter Sikachev, Andreas H. König

June 16, 2011

Contents

1 Introduction

Enhancing the visual appearance of plain objects by applying definitions of fine structures to surfaces is called *Texturing* [3, 4]. Different material properties may be simulated:

- Color
- Reflection
- Gloss

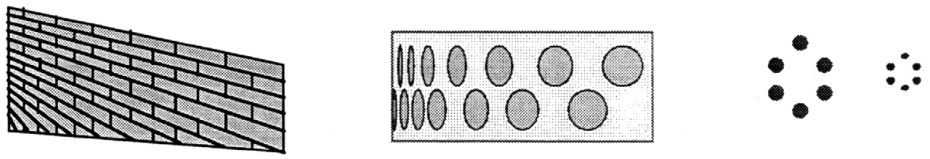

Figure 1: Texture deformations: size, shape, and density.

- Transparency
- Bump

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. The procedure of applying a texture to an object is called *mapping*. Two steps are used to define the mapping: *texture coordinates (u, v)* are mapped into *object space coordinates* $(\overline{x}, \overline{y}, \overline{z})$, 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 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 $(\overline{x}, \overline{y}, \overline{z}) = O(u, v)$ be the mapping from texture space to object space (respectively $\overline{x} = O_{\overline{x}}(u, v)$ and $\overline{y} = O_{\overline{y}}(u, v)$). 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 $S(x, y) = T(O^{-1}(\overline{x}, \overline{y}, \overline{z}))$ has to be found.

The parametrizaton O defines the mapping 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.

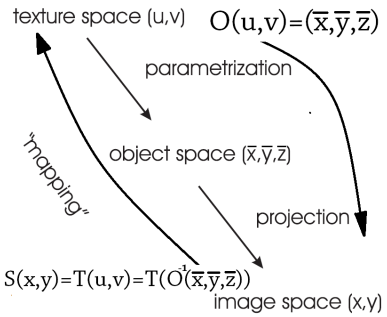

Figure 2: Texture mapping.

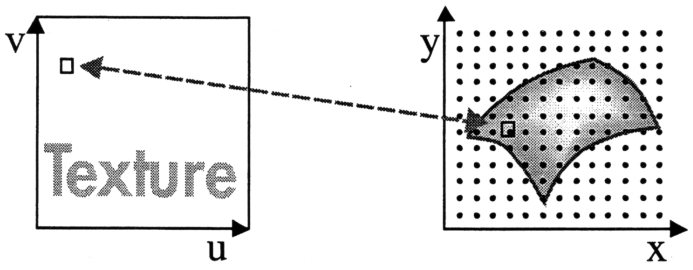

Figure 3: Texture space vs. image space.

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 barycentric coordinates: $(\overline{x}, \overline{y}, \overline{z}) = a_0 * (\overline{x_0}, \overline{y_0}, \overline{z_0}) + a_1 * (\overline{x_1}, \overline{y_1}, \overline{z_1}) + a_2 * (\overline{x_2}, \overline{y_2}, \overline{z_2})$. (a_0, a_1, a_2) are the barycentric coordiantes. The according texture coordinates (u, v) for point $(\overline{x}, \overline{y}, \overline{z})$ are 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).

A parametrization for patch-surfaces is realized in a straight forward way by utilizing the patch parameters as texture coordinates (u, v) (refer to Fig-

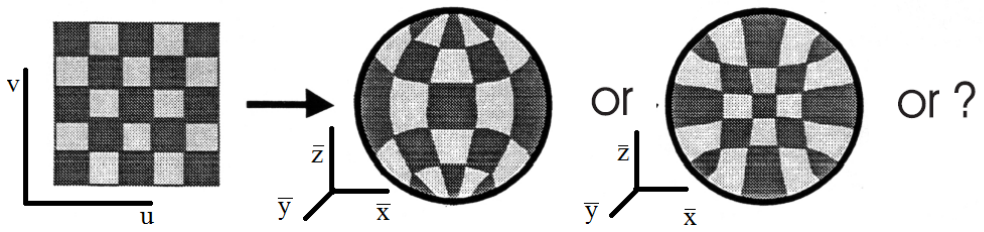

Figure 4: Definition of parametrization.

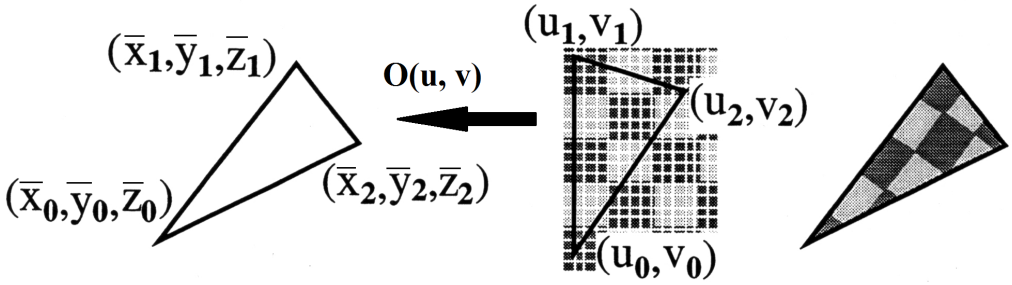

Figure 5: Parametrization for a triangle.

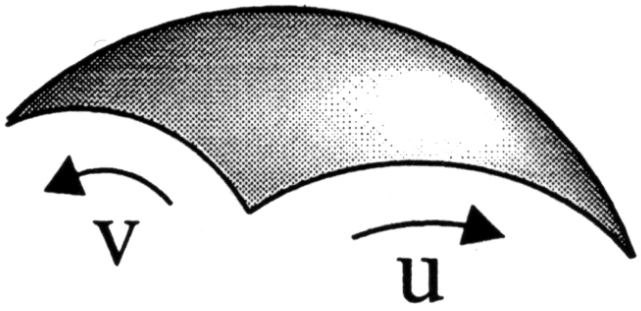

Figure 6: Parametrization of a patch.

ure 6).

2 Type 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 [6].

2.1 Scanline oriented mapping

Scanline oriented rendering methods are calculating images line by line [12]. With the usage of textures, the inverse mapping function $S(scanline)$ is needed. If S is not available, the texture itself has to be dealt with in a line by line way. This technique is referred to as texture scanning. It may result in holes in the calculated image or multiple evaluations of the same image

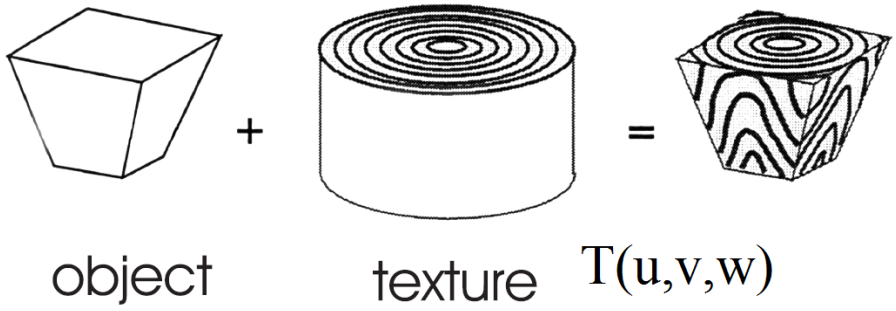

Figure 7: Pyramid cone with solid wood texture.

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.

2.2 Procedural and solid texturing

Three-dimensional textures are usually defined as a function $T(x, y, z)$ in object space [9, 10]. Nevertheless it is possible to specify the texture in 3D-parameter space like $T(u, v, w)$. This approach eliminates the parametrization step, yielding an undeformed texture. This type of texturing, when a texture is generated in a procedure instead of sampling a raster image, is called *procedural texturing*. *Solid texturing* is a process where the texture generating function is evaluated over R^3 at each visible surface point of the model.

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

$0 \leq \overline{x}^2 + \overline{y}^2 \leq 1, \overline{z}\ arbitrary \rightarrow lightbrown$
$1 \leq \overline{x}^2 + \overline{y}^2 \leq 2, \overline{z}\ arbitrary \rightarrow darkbrown$
$2 \leq \overline{x}^2 + \overline{y}^2 \leq 3, \overline{z}\ arbitrary \rightarrow yellow$
etc.

Figure 7 shows the influence of this texture on an object. Another approach of procedural texturing which results in a solid texture is called bombing. A number of spheres with random size and location is defining an appearance of the texture similar to a swiss cheese. Figure 8 shows an example.

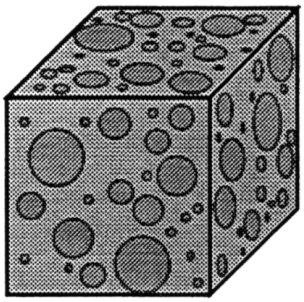

Figure 8: Bombing.

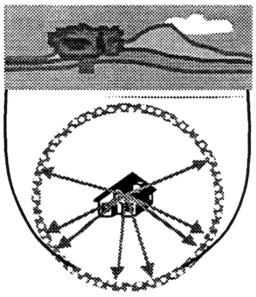

Figure 9: Environment mapping.

2.3 Environment mapping

With this approach, the texture is defined as a surrounding environment, which is shrink-wrapped to the object [1]. In a preprocessing step, an environment map is defined. The desired texture is projected to a surrounding hull or sphere of the object. As the surrounding sphere is much larger as the object to be mapped to, it is 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 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 into account. Raytracing can be used to generate the map for the sphere, where just viewing rays are considered,

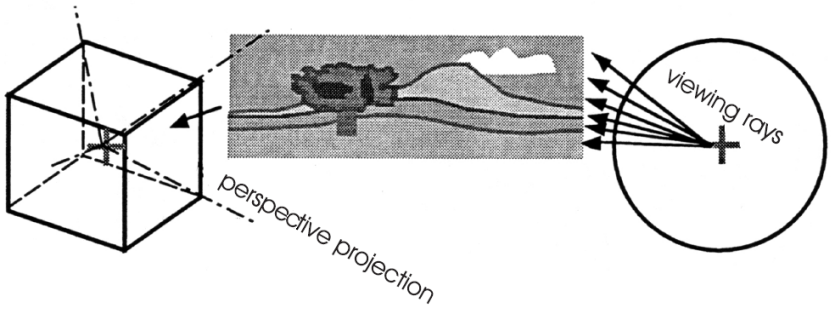

Figure 10: Environment map generation for cube and cylinder maps.

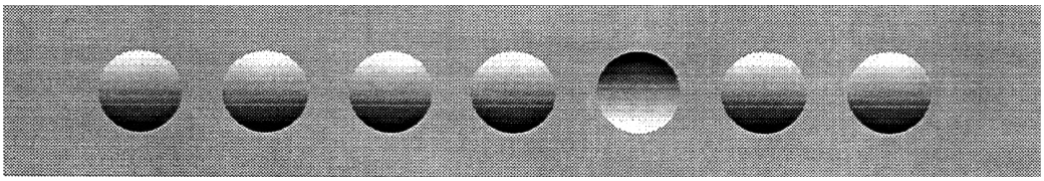

Figure 11: Bump mapping.

but no reflected rays. Figure 10 shows the difference.

2.4 Bump mapping

Bump mapping is able to simulate minor imperfections of surfaces [2]. As modeling the bumps geometrically would be too resource consuming, the bumps are just simulated by shading the surface a little differently. Figure 11 shows the principle.

As the lightness of a certain point depends 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.

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.

2.5 Horizon mapping

Horizon mapping tries to simulate bumps in a more sophisticated way than standard bump mapping [8]. For each texel eight horizon values are calculated, which are interpolated before shading. A point in object space is only

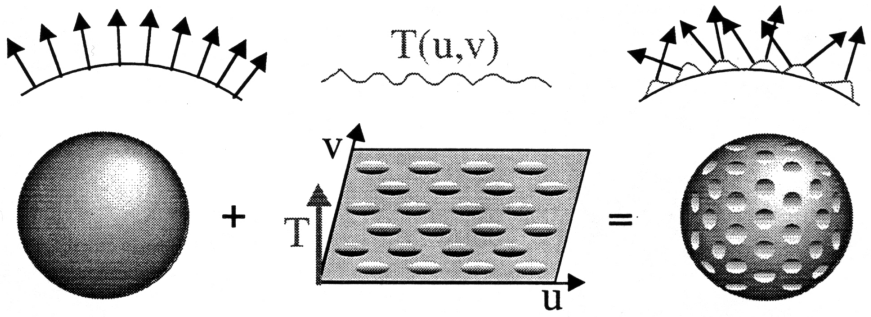

Figure 12: Bump mapping.

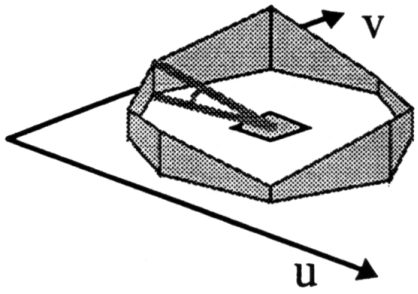

Figure 13: Horizon mapping.

directly lit, 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. With horizon mapping two visual effects are possible that are not available with bump mapping: bumps are casting shadows and bumps just below the shadow horizon are not falsly lit.

2.6 Parallax mapping

Parallax mapping is implemented by displacing the texture coordinates at a point on the rendered polygon by a function of the view angle in tangent space (the angle relative to the surface normal) and the value of the height map at that point [7]. At steeper view angles, the texture coordinates are displaced more, giving the illusion of depth due to parallax effects as the view changes, as shown in Figure 15.

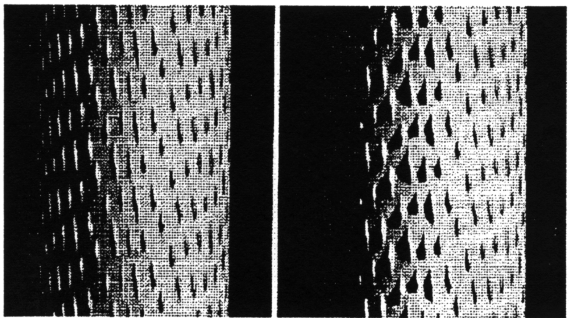

Figure 14: Two cylinders rendered with horizon mapping (right hand side) and without (left hand side).

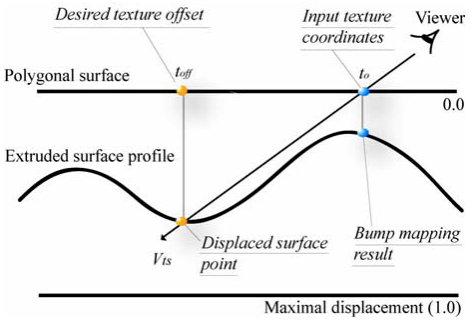

Figure 15: Displacement based on sampled height field and current view direction.

2.7 Relief mapping

Relief mapping is an alternative texture mapping technique to parallax mapping that is much more accurate, and can support self-shadowing and normal mapping [11]. It can be quite simply described as short-distance raytracing done in a pixel shader (shown in Figure 16). Various techniques can be implemented to speed up the raytrace by taking variable step sizes. Relief texture mapping supports the representation of 3D surface detail, producing self-occlusion, self-shadowing (see Figure 17), view-motion parallax, and silhouettes.

3 Perspective correctness

Texture coordinates are specified at each vertex of a given triangle, and these coordinates are interpolated using an extended Bresenham's line algorithm. If these texture coordinates are linearly interpolated across the screen, the

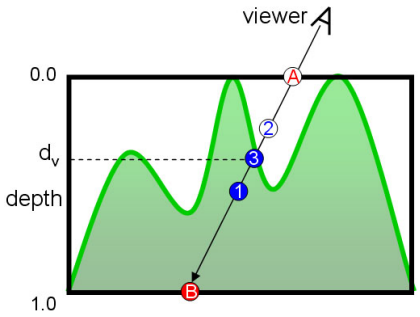

Figure 16: Ray intersection with a height-field surface using binary search. Starting with A and B, the numbers indicate the sequence of computed midpoints.

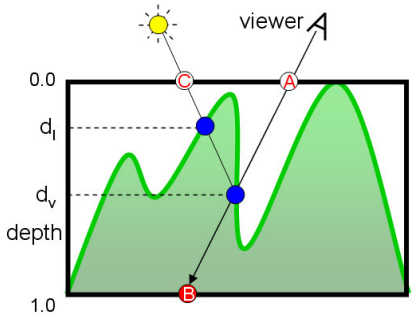

Figure 17: Shadow computation. One needs to decide if the light ray intersects the height-field surface between point C and the point where the viewing ray first hits the surface.

result is affine texture mapping. This is a fast calculation, but there can be a noticeable discontinuity between adjacent triangles when these triangles have different angles to the view plane. In Figure 18 center the texture (the checker boxes) appear bent.

Perspective-correct texturing accounts for the vertices' positions in 3D space, rather than simply interpolating a 2D triangle. This achieves the correct visual effect, but it is slower to calculate. Instead of interpolating the texture coordinates directly, the coordinates are divided by their depth (relative to the viewer), and the reciprocal of the depth value is also interpolated and used to recover the perspective-correct coordinate. This correction causes that in parts of the polygon which are closer to the viewer the difference from pixel to pixel between texture coordinates is smaller (stretching the texture wider), and in parts which are farther away this difference is larger (compressing the texture).

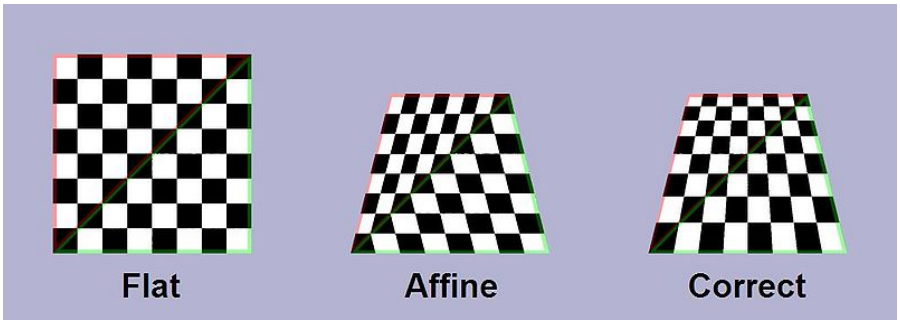

Figure 18: Affine texture mapping does not take into account the depth information of a polygon's vertices. If a polygon is not perpendicular to the viewer it produces a noticeable defect.

Affine texture mapping directly interpolates a texture coordinate u_α between two endpoints u_0 and u_1:

$$u_\alpha = (1-\alpha)u_0 + \alpha u_1; 0 \leq \alpha \leq 1$$

Perspective-correct mapping interpolates after dividing by depth z, then uses its interpolated reciprocal to recover the correct coordinate:

$$u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}} \tag{1}$$

4 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 19 shows one of the problematic regions when mapping a texture to a sphere.

The reason for aliasing is to be found in the mapping process due to the parametrization and/or projection. If the projection of an image-space pixel covers more than one texel in texture space, it is not an appropriate strategy to choose just one of the covered texels for color evaluation. A better way to do it would be to use a weighted average of the covered texels. Figure 20 depicts the mapping situation. This averaging can either be done during texture evaluation (*direct convolution*) or in a preprocessing step (*prefiltering*).

Figure 19: Very fine structures at the pole of the sphere will yield artifacts.

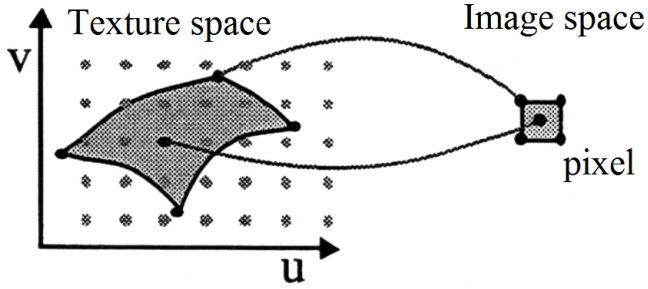

Figure 20: The origin of artifacts is to be found in the mapping procedure. If more than one texel is covered by the projected pixel, a weighted average of the texels should be used.

4.1 Direct convolution

Due to the limited time constraints during rendering, just approximations of the projected pixels are used. Simple shapes like rectangles of ellipsoids are evaluated for generating the average of the covered texels. Figure 21 gives examples. This approach may be computationally expensive, yet it yields superior quality, as nearest neighbour sampling.

4.2 Mip-mapping

A more efficient method is the prefiltering approach called mip-mapping [13]. In a preprocessing step different resolutions of the texture are precalculated, each reduced version being half the size in each dimension. The coarsest representation features a single texel. Figure 22 shows the reduction process. The prefiltered textures can be efficiently stored illustrated on a scheme, which separates the RGB channels, shown in Figure 23. The additional storage requirement is one third of the initial texture. In order to map the

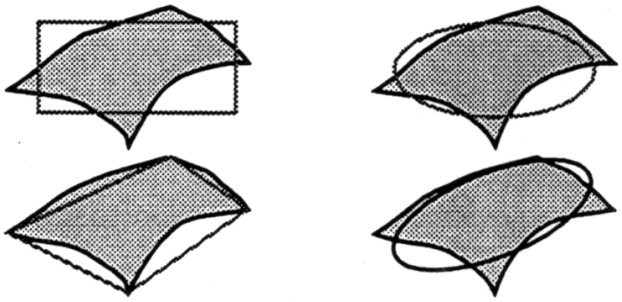

Figure 21: Direct convolution: simple shapes are used to approximate the averaging of the covered area.

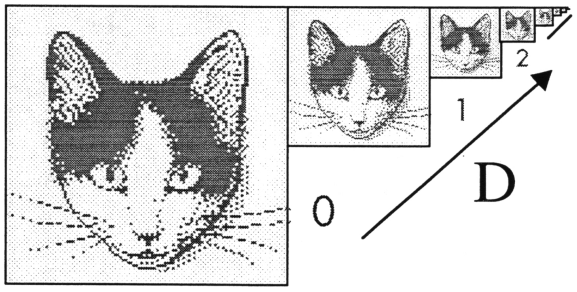

Figure 22: Mip-mapping texture downsampling.

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 approximately calculate this level: $D = ld(max(|d_1|, |d_2|))$ (see Figure 24). Usually D will not be an integer number, therefore interpolation between the according smaller and larger maps is done:

$$color\_of\_pixel = (D_1 - D) * W_0 + (D - D_0) * W_1 = (D_1 - D) * (W_0 - W_1) + W_1$$

where W_0 is the value from map $D_0 = trunc(D)$ and W_1 is the value from map $D_1 = D_0 + 1$, derived by linear interpolation. If the original resolution is too small ($D < 0$), the texture has to be enlarged (*upsampling*) by interpolating the finest resolution map. A better quality cannot be achieved, as the texture is not defined more accurately (refer to Figure 25):

$$T(u + du, v + dv) = du * dv * T(u + 1, v + 1) + du * (1 - dv) * T(u + 1, v) + (1 - du) * dv * T(u, v + 1) + (1 - du) * (1 - dv) * T(u, v)$$

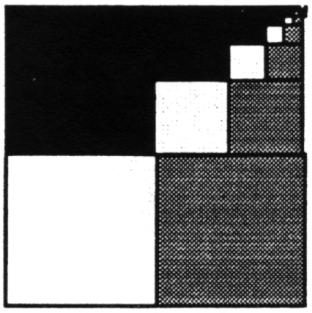

Figure 23: Mip-mapping texture storage scheme.

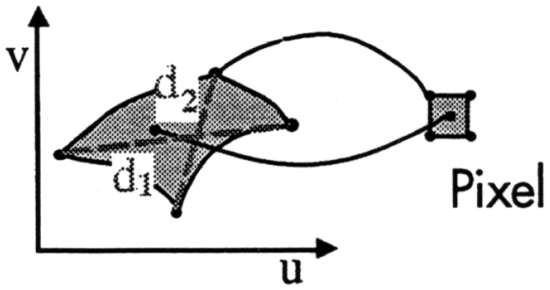

Figure 24: Mip-mapping texture level derivation by length of diagonals.

4.3 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 26) [5]:

$$S(u_0, v_0) = \sum_{u \leq u_0, v \leq v_0} T(u, v)$$

This allows to approximate texture footprints with corresponding axis-aligned rectangular areas (see Figure 21). Whenever a sum of a certain rectangular region in texture space is 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 four memory accesses and additions are necessary to compute the sum. The obviuos drawback of the method is decreasing precision for high-resolution textures (when the sum becomes too big to be stored in the texel). Figure 27 shows an example of summed area table usage.

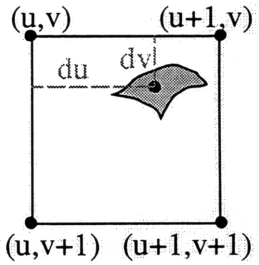

Figure 25: Mip-mapping texture enlargement interpolation.

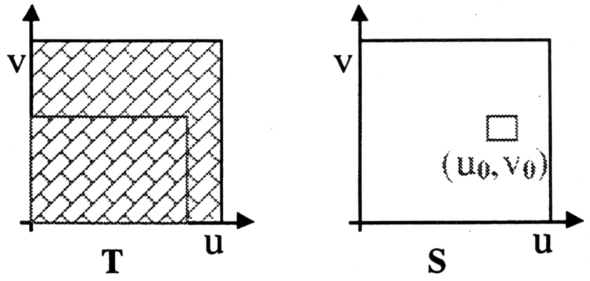

Figure 26: Summed area table generation.

References

[1] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19:542–546, 1976.

[2] James F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pages 286–292, aug 1978.

[3] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, Dept. of CS, U. of Utah, Dec 1974.

[4] Edwin E. Catmull. Computer display of curved surfaces. In *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structures*, pages 11—17, May 1975.

[5] Franklin C. Crow. Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph.*, 18:207–212, January 1984.

[6] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan. Volume rendering. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 65–74, 1988.

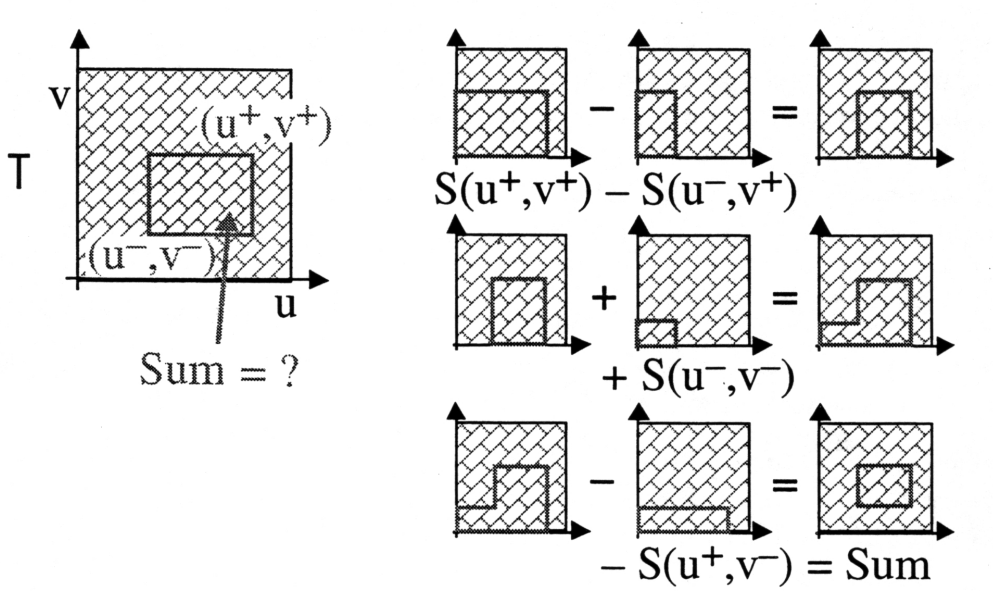

Figure 27: Summed area table access.

[7] Tomomichi Kaneko, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda, and Susumu Tachi. Detailed shape representation with parallax mapping. In *Proceedings of the ICAT 2001*, pages 205–208, 2001.

[8] Peter-Pike Sloan Michael and Michael F. Cohen. Interactive horizon mapping. In *Rendering Techniques 00 (Proc. Eurographics Workshop on Rendering)*, pages 281–286. Springer, 2000.

[9] Darwyn R. Peachey. Solid texturing of complex surfaces. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 279–286, jul 1985.

[10] Ken Perlin. An image synthesizer. In *ACM SIGGRAPH 1985*, volume 19, pages 287–296, New York, NY, USA. ACM.

[11] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *ACM SIGGRAPH 2005*, volume 24, pages 935–935, New York, NY, USA. ACM.

[12] Turner Whitted. A scan line algorithm for computer display of curved surfaces. In *ACM SIGGRAPH 1978*, volume 12, pages 8–13, New York, NY, USA. ACM.

[13] Lance Williams. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, volume 17, pages 1–11, jul 1983.

[14] C. Wylie, G. W. Romney, D. C. Evans, and A. Erdahl. Halftone perspective drawings by computer. In *Proc. AFIPS FJCC 1967*, volume 31, 1967.