Real-Time Rendering Cpt 6

ianaesthetic

October 5, 2017

6 Texture

6.1 The Texturing Pipeline

Texturing, at its simplest, is a technique for efficiently modeling the surface's properties. The pixels in the image texture are called *texels.* The gloss texture modifies the gloss value, and *bumping texture* changes the direction of the normal, which will all influence lighting equation.

$$\begin{aligned}
&\textit{object space location} \xrightarrow{projector\ function} \textit{parameter space coordinate}\\
&\xrightarrow{corresponder\ function} \textit{texture space coordinate}\\
&\xrightarrow{obtain\ value} \textit{texture value}\\
&\xrightarrow{valuetransformfunction} \textit{transformed texture value}
\end{aligned}$$

This projector function is called *mapping*, which needs to *texture mapping.*

6.1.1 The Projector Function

Projector functions typically work by converting a three-dimensional point in space into texture coordinates, including spherical, cylindrical, planar projections. Problems may occur at the seams where faces meet. *Polycube map* maps a model to a set of cube with different volumes of space mapping to different cubes. Other form of projector functions are not projections but are implicit part of surface formation. The goal of the projector function is to generate texture coordinate.

Various projector functions can be applied to a single model. most projector function are applied in modeling stage and results are stored in vertices. Some functions require vertex or pixel shader like animation and *environmental mapping.*

Spherical projector function, according to spherical coordinates:

$$\phi(x,\ y,\ z) = (\frac{\pi + \mathbf{atan2}(y,\ x)}{2\pi}, \frac{\pi - \mathbf{acos}(\frac{z}{\|x\|})}{\pi})$$

Cylindrical:

$$\phi(x,\ y,\ z) = (\frac{\pi + \mathbf{atan2}(y,\ x)}{2\pi}, \frac{1}{2}(1+z))$$

Planar projection simply uses orthogonal projection to apply texture maps to characters, as the texture glued onto a paper doll.

$$\phi(x,\ y,\ z) = (\frac{\tilde{u}}{w}, \frac{\tilde{v}}{w}), \quad \text{where} \begin{pmatrix} \tilde{u} \\ \tilde{v} \\ * \\ w \end{pmatrix} = \mathbf{P}_t \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Artists often must manually decompose the model into near-planar pieces to avoid distortion by unwrapping the mesh. The goal is to have each polygon be given a fairer share of a texture's area, while also maintaining mesh connectivity (without too many seams).

The parameter space is not always a two-dimensional plane; sometimes is a three-dimension volume. For three-element vector (u, v, w), w represents the depth along the projection direction. For four-coordinate (s, t, r, q), q is used as fourth value in a homogeneous coordinate for spotlighting effect. Another important type of parameter space is directional, where each point in the parameter space represents a direction. The most common one is *cube texture*. The one-dimensional projector function also have its own use(coloration according to altitude). Line can also be textured.(Rain is textured with a semi-transparent image)

6.1.2 The Corresponder Function

Corresponder function convert parameter-space coordinate to texture-space location in order to provide flexibility. One type is to use the API to select a portion of existing texture for display. Another type is a matrix transformation, which can be applied in vertex or pixel shader. The order of the transformation must be reserved as it's the location of image will be changed rather the location of object.

Another class of corresponder functions controls the way an image is applied when parameters are out of range $[0, 1)$.

Warp(DirectX), **Repeat**(OpenGL), or**title**: The image repeats itself across the surface; algorithmically, the integer part will drop.

mirror: The image repeats itself across the surface, but is mirrored on every other repetition. This provide some continuity along the

Clamp(DirectX), **Clamp to edge**(OpenGL): Values out of range of$[0, 1)$ will clamp to the edge. This function is useful for avoiding accidentally taking samples from the opposite edge of a texture when bilinear interpolation happens near a texture's edge.

Border(DirectX), **Clamp to Border**: Values out of range of $[0, 1)$ will be set to a border color. This function is good for rendering decal onto surfaces.

The *periodicity* problem like repeating tiles of a texture can be solved by combining the texture values with another, non-tiled texture. Another option to avoid periodicity is to use shader programs to implement specialized corresponder functions that randomly recombine texture pattern or tiles. *Wang tiles* are one example which choose randomly from small square tiles with matching edges.

For real-time work, the last corresponder function is implicit, and is derived from the image's size. It allows the parameter being in range $[0, 1)$ with different resolution of texture can be applied.

6.1.3 Texture Value

The texture value is retrieved after corresponder function. For image textures, this is done by using the texture to retrieve texel information from the image. Procedural functions are sometimes used.

The most common texture values include RGB triplet, gray scale and RBGα value. The values returned from the texture are optionally transformed before use. One common example is the remapping of data from unsigned range to a signed range. Another one is to compare the value with a reference value and return a flag.

6.2 Image Texturing

The texture used in GPU has usually $2^m \times 2^n$ texels. Modern GPU allows for an arbitrary size. When finally projecting texture onto the screen, the projected square may contain *magnification* and *minification.* The filtering can take place in the input(values read from textures) or in the output(final pixel colors). When input and output is linear, there will be no difference.

6.2.1 Magnification

The most common filtering techniques for magnification is *nearest neighbor*(application of box filter) and *bilinear interpolation.* There is also *cubic convolution*, enabling high quality. It's not commonly available in the native hardware but can be realized in shader program.

One characteristic of nearest neighbor is that the individual texels may become apparent, which is called *pixelation* for every pixel only fetch one nearest texel to each pixel center.

In bilinear interpolation, for each pixel, this kind of filtering finds the four neighboring texels and linearly interpolates in two dimension to form a blended value for pixel. The cubic convolution is more expensive. Specifically, for a image coordinate (p_u, p_v). According to the coordinate, four centers of texels can be calculated.

$$\begin{aligned} x_l &= \lfloor p_u \rfloor, \quad y_b = \lfloor p_v \rfloor \\ x_r &= \lceil p_u \rceil, \quad y_t = \lceil p_v \rceil \end{aligned}$$

When interpolating:

$$u^{'} = p_u - \lfloor p_u \rfloor, \quad v^{'} = p_v - \lfloor p_v \rfloor$$

$$\begin{aligned} \mathbf{c}(p_v,\ p_u) = &(1-u^{'})(1-v^{'})\mathbf{t}(x_l,\ y_b) \\ &+ u^{'}(1-v^{'})\mathbf{t}(x_r,\ y_b) \\ &+ (1-u^{'})v^{'}\mathbf{t}(x_l,\ y_t) \\ &+ u^{'}v^{'}\mathbf{t}(x_r,\ y_t) \end{aligned}$$

This kind of interpolation will have poor performance when the image has a nonlinear change such as the edge of a object.

A common solution to the blurriness that accomplish magnification is to use *detail textures.* These are textures that represent fine surface details. The high-frequency of repetitive pattern of the detail texture, combined with low-frequency magnified texture will have a similar visual effect with high resolution texture.

Sometime, linear interpolation is not required. In order to magnify checkerboard, remapping will have better visual effect. Such remapping is useful in cases where well-defined edges are important, such as text. Alpha value can be used to be threshold the results. This method is called *cutout texture. Vector* *textures* can handle high quality edges generation. It stores information at texel describing how the edges cross the grid. These information can contain more subjects leading to accuracy and resolution-independence. However, it's too costly to apply to read-time rendering application.

There is simpler technique using *sampled distance filed* data structure. A distance filed is a scalar signed field over a space or surface in which an object is embedded(that means, the filed can be curved). At any point, the distance field has a value with a magnitude equal to the distance to the object's nearest boundary point. The value is negative within the object and positive outside the object. This will interpolated with the real distance rather than space coordinate so it's linear and fit bilinear interpolation to get a sampled distance filed for further rendering. The evaluation of sampled distance field is to use alpha map and built-in GPU interpolation and alpha testing rather than pixel shader modification.

6.2.2 Minification

Directly, we consider how to aggregate the influence of texels, which is hard to accomplish.

Nearest neighbor can be used to use one texel at the very center of the pixel to represent, which will cause big alias and even noticeable when surface is moving and result *temporal aliasing.* Similarly, bilinear interpolation can be used but don't perform much better than nearest neighbor.

According to Nyquist, we need to increase the pixel sample rate or the the texture frequency has to decrease. The previous chapter introduces increasing of sampling rate, but has a limited increase that don't fit minification. The design of anti-aliasing algorithm is to pre-process texture and create data structure to help with effect approximation.

The most popular method is called *mipmapping.* Mipmapping generate a set of images, called *mipmap chain*, with a level start at 0 and each level downsample to a quarter of the original area as the next level. level 0 is the original one and the downsampled ones are called subtexture.

Two important elements in forming high-quality mipmaps are good filtering and gamma correction. The common way to form a mipmap level is to take each 2×2 sets of texels and average them to get the mip level texel. This is actually a 2-dimensional box filter, which is better to use a Gaussian filter instead.

For textures encoded in a non-linear space (such as color texture), gamma correction is obviously needed to apply to the texture before mipmapping to a linear space. After filtering, converting texture back to non-linear space is also needed.

For the texture is fundamentally in a non-linear space, specialized mipmapping method is required.

In order to measure the coverage of a single pixel, d or λ is introduced. There two approaches to compute it: First we use the longest edge of quadrilateral formed by the pixel's cell to approximate; Or we can select the largest absolute value of $\frac{\partial u}{\partial x}$, $\frac{\partial v}{\partial x}$, $\frac{\partial u}{\partial y}$, $\frac{\partial v}{\partial y}$. Each differentials is a measure of the amount of change in the texture coordinate with respect to a screen axis. These gradient value is not accessible by dynamic flow control. And as the vertex shader can't access derivative data so it needs to be calculated first and then sent into GPU.

d is used to determine where to sample along the mipmap's pyramid axis. The goal of texel ratio is at least 1:1 according to Nyquist. The important principle is that the more texels a pixel covers and d increases, a smaller and blurrier version of texture will be accessed by a triplet (u, v, d). Instead of a integer representing mipmap level, d is a fractional value between two level. Using u, v coordinates we can get two bilinear texture values by bilinear interpolation and linear interpolated them using the decimal part of d to get the final value. This process is called trilinear interpolation and is performed in each pixel.

Level of detail bias(LOD bias) is used to control by being added to d in different circumstances. It can be specified for the texture as whole, or per-pixel in the pixel shader.

Instead of summing all the texels that affect a pixel, pre-combined set of texels are accessed and interpolated. It may lead to *overblurring*, which happens when camera looks along nearly edge-on. It may have a rectangular area. In order to avoid aliasing, we may use the largest derivative as d and result in relatively blurring. One extension to solve this is *ripmap*, which will also create rectangular texture. However, it's too costly to use in real-time rendering.

Another method to avoid overblurring is *summed-area table*(SAT). Every texel has more bits of precision and record its individual data(color for example) and the sum of all the corresponding texture's texels in the rectangle formed by this location and texel (0, 0). When a pixel grid is back-projected (which means project the pixel grid to the texture space; normally, we perceive as the texture projected on the pixel space on the screen) onto the texture, the projected gird is bounded by a rectangular. The average data in the rectangular will be returned as the texture value.

$$\mathbf{c} = \frac{\mathbf{s}(x_{ur}, y_{ur}) - \mathbf{s}(x_{ll}, y_{ur}) - \mathbf{s}(x_{ur}, y_{ll}) + \mathbf{s}(x_{ll}, y_{ll})}{(x_{ur} - x_{ll})(y_{ur} - y_{ll})}$$

However, both ripmap and summed-area table will have blurring image when a texture is viewed along the diagonal as the average of the whole texture will be averaged rather than the average the thin real projected rectangle.

Ripmap and summed-area table are examples of *anisotropic filtering* algorithm. They have good effect in horizontal and vertical directions but is memory intensive. Summed-area table are more suitable in modern GPU with more algorithms to optimize.

Other from anisotropic filtering, *unconstrained anisotropic filtering* is widely used. Similar to summed-area table, pixel grid is back-projected to a quad and sampled a number of time with associated squarish area. Instead of using a single mipmap sample to approximate the coverage, the algorithm uses a number of squares to cover the quad. The shorter side of the quad is used to determine d with a relatively small averaged are for each sample. The longer side creates a *line of anisotropy* parallel to the longer side and through the middle of the quad. Samples will be taken on the line and its amount is decided by the value of anisotropy.

This scheme allows the line of anisotropic to run in any direction (Correctly sample the diagonal view of texture as the line can be diagonal rather than horizontal and vertical in common anisotropic filtering). Meanwhile it has same cost as the mipmap does.

6.2.3 Volume Textures

A direct extension of image textures is three-dimensional image data that is accessed by (u, v, w) value. Being inside A single mipmap level requires a trilinear interpolation, while filtering between mipmaps require *quadrilinear interpolation.* This requires for high precision texture. Volume textures have the advantage that the good projection function which avoids distortion and seam.

6.2.4 Cube Maps

Another type of texture is the *cube texture* and *cube map.* The mapping point is specified by a three-dimensional vector. It points from the center of the cube and choose the texture face by the face with largest magnitude. (choose $-Z$ for $(1, 2, -3)$. The other entries will be divided by magnitude to be in range in [-1, 1] and can simply be mapped to range [0, 1]. It's useful for environmental mapping.

Cube maps support bilinear interpolation and mipmaps, but may have problems near the seam as the sample can't be across the boundaries of faces and faces can't affect each other. Another problem that the angular size of a texel varies over a cube face. That is, a texel at the center of a cube represents a greater visual solid angle than a texel at the corner.

6.2.5 Texture Caching

Memories for textures are always not enough. *Texture Caching* aims to a balance between the overhead of uploading textures to memory and the amount of of memory taken up by textures at one time.

Some general advice is to keep the texture no larger than necessary and to try to keep polygons grouped by their use of texture.

A *least recently used*(LRU) strategy is one commonly used in texture caching scheme. Every texture has a time stamp representing when it's last accessed. When new texture is uploaded, the texture with oldest time stamp is unloaded first. Priority is optionally given to prevent unnecessary texture swapping.

It's suggested to check the texture being swapped out for problems will happen when this texture is used in current frame. When it occurs, LRU has terrible performance. It better to first transfer to *most recently used* and after no textures are swapped out to switch back to LRU.

Prefetching where future needs are anticipated making the texture loading over a few frames as there may be sudden big amount of texture loading and it takes a lot of time.

When data set is huge, such as flight simulations program. The traditional approach is to break these images into smaller tiles that hardware can handle. A improved structure is *clipmap.* The entire data set is treated as a mipmap, but only a small part of the lower levels of the mipmap is required for certain view and memory.

6.2.6 Texture Compression

Texture Compression can directly attack memory and bandwidth problem. *S3 texture compression* becomes the standard for DirectX and called *DXTC* or *BC.* It has independently encoded pieces on 4×4 texel blocks, which is also called *tile*. Encoding is based on interpolation which is simple and fast. There are five variants:

DXT1 / BC1: It has two 16-bit reference RGB565 values as the limitation. Each texel has a 2-bit interpolation factor to refer to 2 references or 2 intermediate value, while only 1 intermediate value is included and another stands for transparent when alpha is introduced. The compression ratio is 6 : 1 compressing 24-bit RGB texture.

DXT3 / BC2: The encoding pattern is same as DXT1. Meanwhile, every texel has a 4-bit alpha value stored separately. The compression ratio is 4 : 1.

DXT5 / BC3: The encoding pattern is same as DXT1. In addition, alpha data is encoded using two 8-bit reference values. Each texel has 3 interpolation factor which can refer to one of the reference alpha values or one of the six intermediate alpha values.

ATI1/ BC4: It supports single color channel and encoded as the alpha channel in DXT5.

ATI2/ BC5: It supports dual color channels and encoded as the aloha channel in DXT5.

The disadvantage of these compressions is *lossy*. Once a tile has many distinct value it will lead to some loss. Fortunately, These compression scheme generally give acceptable image fidelity. Another problem is that all the colors are on a straight line in RGB space. *color distribution* can attack it as it uses colors from the neighbors blocks in order to achieve more colors.

For OpenGL ES, *Ericsson texture compression(ETC)* is chosen. It shares the same features as S3TC. It encodes 4×4 texels with 4 bits per texel. Each 2×4 or 4×2 block stores a basic color, and each texel in a block can select to add one of the values in the selected lookup table.

Compression of normal maps requires some care. We assume the normal is unit vector and the z-component is positive. So we can derive z by:

$$n_z = \sqrt{1 - n_x^2 - n_y^2}$$

So the three-dimensional texture can be stored as two-dimensional texture, a modest compression. Further compression is usually achieved in BC5 manner: A block of texel has 16 possible values, which can be bounded by a bounding box. So x, y values can be perceived as two channels of color and interpolation factor allows for value to choose from for each axis.

A fallback when GPU does not support BC5 can use DXT5 to substitute.

For normal maps, the aspect ratio decides the layout of the normal inside a block. For example, as the normal has a width nearly twice as height, the current 8×8 grids can be converted to 4×16 grids with different bits allocation.

6.3 Procedural Texturing

Procedural texture is to evaluate a function to look up texture value rather than generate texture space coordinate. It's often used in offline rendering system and not so common in real-time rendering. Procedural texture is used when image texture requires for costly memory access. Volume texture is a typical example.

Cellular texture, calculated the closest distance to a set a special points for every location. The color or shading normal will change by this distance.

Another type of application is on physical simulation like water ripples.

As parameterization will be more difficult for procedural texture, it's better to synthesis texture onto the surface directly. Anti-aliasing procedural texture becomes both easier and more difficult. On one hand, pre-computations methods are not available; On other hand, procedural texture have more inside information.

6.4 Texture Animation

The image applied and coordinate can both be dynamic. For example, dynamic coordinate for water downfall to make the water move. The texture can be applied to matrix transformation and blending techniques.

6.5 Material Mapping

A common use of a texture is to modify a material property affecting the shading equation. Shader can read values from the texture directly. For equation:

$$\mathbf{L}_o(\mathbf{v}) = (\frac{\mathbf{c}_{\text{diff}}}{\pi} + \frac{m+8}{8\pi}\overline{\cos}^m \theta_h \mathbf{c}_{\text{spec}}) \otimes E_L \overline{\cos}\,\theta_i$$

$\mathbf{c}_{\text{diff}}$, $\mathbf{c}_{\text{spec}}$ and m is the material information can be read from textures, particularly *diffuse color map, specular color map*(grayscale value rather than RGB for the reality) and *gloss map*(describing the smoothness). As $\mathbf{c}_{\text{diff}}$, $\mathbf{c}_{\text{spec}}$ is linear to the output, the input can be filtered without anti-aliasing, when m needs some care.

6.6 Alpha Mapping

The alpha map can be used for many interesting effect like decaling with clamp corresponder function and transparent edge to present the part you want. A similar application is in making cutouts that have 3D visual effect like cross tree. In order to reserve the visual effect from other angle, cross tree is set by rotating the cutouts. As illusion will break down when viewer see from the above, more cutouts can be added. Combining alpha texture and texture animations can make many visual effects.

Alpha map has many options like alpha blending. As discussed before, alpha blending needs a relative order. *Alpha test* is another option that conditionally discarding pixels with alpha values below a given threshold in the merge unit or pixel shader, which enable polygons to be rendered in any order.

Alpha to coverage, and the similar feature *transparency adaptive anti-aliasing*, take the transparency value of the fragment and convert this into how many samples inside a pixel are covered. This is similar to screen-door transparency but at a sub-pixel level. The alpha value determines the proportion of samples being covered within a pixel.

6.7 Bump Mapping

Bump mapping is used to represent small-scale detail which is implemented in per-pixel shading. The scale of detail on object can be classified into: *macro-features*, *meso-feature* and *micro-features*

Macro features cover many pixel. Macro-geometry is represented by geometric primitives, while micro-geometry is encapsulated in the shading model and used in pixel shader using texture maps as parameters for equation. Meso-geometry describe everything between these two scales.

Bumping mapping techniques are commonly used fro mesoscale shading, adjusting the shading parameters(like normals) at pixel level in such a way that the viewer perceives small perturbations away from the base geometry. For example, when introducing a change to color component, instead of changing the base texture directly, we access another texture which is used to modify the surface normal. This will preserve the original geometry with different effect.

For bump mapping, the normal must change direction with respect to some frame of reference. *Tangent space basis* is stored per vertex used to transform the lights to a surface location's space to compute the effect of perturbing the normal and is expressed by *tangent and bitangent vectors* and corresponding basis matrix.

$$\begin{pmatrix} t_x & t_y & t_z & 0 \\ b_x & b_y & b_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Tangent and Bitangent don't have to truly be perpendicular to each other since the normal map itself maybe distorted to fit the surface. The normal can be calculated by the product of tangent and bitangent vectors, which can save memory with attention to handedness(e.g, in symmetric model). The handedness can be marked with a bit. The idea of tangent space is important for other algorithms like the orientation of the material.

6.7.1 Blinn's Methods

The original bump mapping method stores two offset values b_u and b_v for interpolated normals along **u** and **v** image axes. This type of of bump map texture is called an *offset vector bump map* or *offset map.*

Another way to represent bumps is to use *heightfield* to represent bumps so the direction of normals.

6.7.2 Normal Mappings

Normal map directly stores perturbed normals, represent the same effect with original bump mapping with different storage format. (x, y, z) values are mapped to [-1, 1], e.g., for an 8-bit texture the x-axis value 0 represents -1.0 abd 255 represents 1.0. It's similar to RGB values and light blue [128, 128, 255] represents flat surface.

Originally, the normal map is in world-space. However, this approach can't deal with any deformation, like rotation, which will lead to change in orientation. So it's rarely used.

Normal map can also be defined in object space. It now can deal with rigid transformation. Lights should be transformed into object space which can be done in application stage.

Tangent space is usually used. It allows for deformation of the surface and maximal usage of normal maps. It can be compressed as introduced before while requiring or more transformations.

There are two methods to use normal maps. If there is seldom lights, it's preferred to transform the light to tangent space as it move slowly per pixel and can be interpolated in triangle. However, it's preferred to transform perturbed normals to world space when there are more lights as it involves less transformation and avoid tangent space distortion.

Filtering normal maps is a difficult problem as the relationship between normal and shaded color is not linear. However, Lambertian surfaces are a special case where the normal map has an almost linear effect on shading. In Lambertian shading, it satisfies:

$$\mathbf{l} \cdot \left(\frac{\sum_{j=1}^{n} \mathbf{n}_j}{n}\right) = \frac{\sum_{j=1}^{n} (\mathbf{l} \cdot \mathbf{n}_j)}{n}$$

So Lambertian shading *almost* produce the right result while it's a *clamped* dot product. But in practice it's not objectionable.

For other surfaces, more details are included in following chapters.

6.7.3 Parallax Mapping

A problem with normal mapping is that the bumps never block each other. The idea of *parallax mapping* is an approximation to shift the pixel location to simulate this effect.

The amount to shift is based on the height retrieved and the angle of the eye to the surface.The heightfield values are scaled and biased before being used. The scale determines how height the heightfield meant to extend and the bias gives a sea-level height at which no shift take place. The equation follows:

$$\mathbf{p}_{\text{adj}} = \mathbf{p} + \frac{h \cdot \mathbf{v}_{xy}}{v_z}$$

Note that unlike most shading equations, here the view vector needs to be in tangent space. The new location also uses the same height as it's about the same. However, this method falls apart at shallow viewing angles. In order to solve this, limited the amount of shifting is needed:

$$\mathbf{p}'_{\text{adj}} = \mathbf{p} + h \cdot \mathbf{v}_{xy}$$

It has some drawbacks, like lessen bumpiness at shallow angles, etc. But it is still considered the practical standard for bump mapping.

6.7.4 Relief Mapping

Parallax mapping is just an approximation. The actual effect we want is what is visible at the pixel, or where the view vector first intersects the heightfield.

Methods are similar to traditional ray tracing with different implementations. Relief mapping is actually a class of three independent research outcomes based on same approaches.

The key idea is to test a fixed number of texture samples along the projected vector in the cut of the surface along the view direction. The algorithm finds the first intersection of the eye ray with the line segments approximating the curved height field. Once the location is determined, the attached map will be used. It can also be used to have the bumpy surface casts shadows onto it self.

The problem of determining the actual intersection point is a root-finding problem. There is a critical importance in sampling the heightfield frequently enough. This can be done either by using better filtering methods or higher resolution of heightfield texture.

Another approach to increasing both performance and sampling accuracy is to not initially sample the heightfield at a regular interval, but instead to try to skip intervening empty space.

One problem with relief mapping methods is that the illusion breaks down along the silhouette edges of objects. The key idea is that the triangles rendered define which pixels should be evaluate by the pixel shader program, not where the surface actually is located. *Shell map* is introduced to extrude each polygon in the mesh outwards and form a prism. Rendering this prism forces evaluation of all pixels in which the heightfield could possibly appear.

6.7.5 Heightfield Texturing

Displacement mapping method has a flat meshed polygon access a heightfield texture and the height retrieved from the texture is used by the vertex shading program to modify the vertex's location. So the heightfield is called *displacement texture*. It's a similar concept to relief mapping. They both have a drawbacks like making collision detection more challenging.