Texturing

Dr. Sheng Bin(盛斌)
Shanghai Jiao Tong University
Lecture 7

What is texturing?

a process that takes a surface and modifies its appearance at each location using some image, function, or other data source.

works by modifying the values used in the shading equations

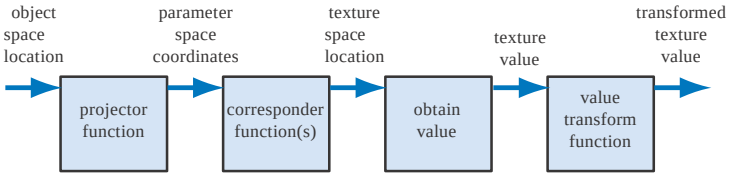
The values changed based on the position on the surface

Texturing. Color and bump maps were applied to this fish to increase its visual level of detail.

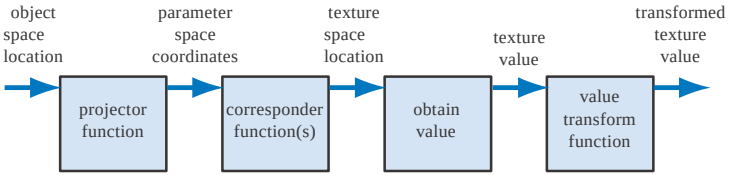
Lecture Overview

- The Texturing Pipeline
- Image Texturing
- Proceduring Texturing
- Some common methods : Material Mapping , Alpha Mapping, Bump Mapping , Parrallax Mapping
- Textured Lights

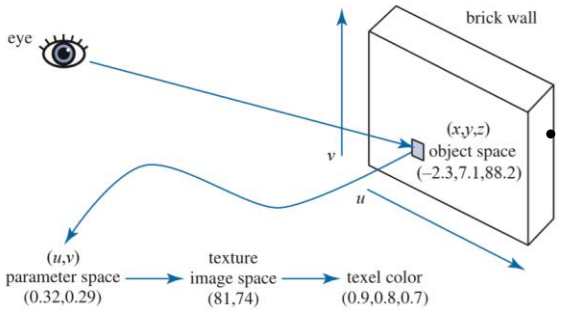
The texturing pipelines



The texturing pipeline



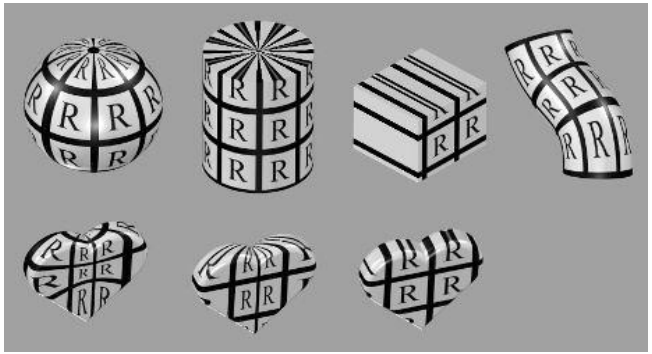
- A location in space is the starting point for the texturing process.
  more often in the model's frame of reference.
- Projector function applies to the point to obtain texture coordinates.
- Corresponder functions can be used to transform the texture coordinates to texture space.
- The retrieved values are transformed by a value transform function, and these new values are used to modify some property of the surface



- a triangle has a brick wall texture and a sample is generated on its surface
- the projector function here typically changes the position (x,y,z) vector into a two-element vector (u,v)
- the corresponder function multiplies the (u,v) by 256 each, giving (81.92,74.24)
- Pixel(81,74) is found in the brick wall image , and is color of (0.9,0.8,0.7)

The projector Function

- converting a 3D point in space into texture coordinates.



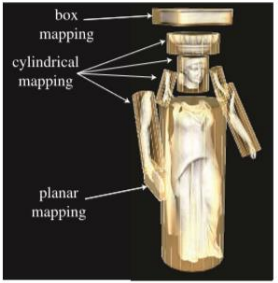
- Different texture projections:
- Spherical, cylindrical, planar, and natural (u,v) projections (from left to right).

- The bottom row : each of these projections applied to a single object (no natural projection).

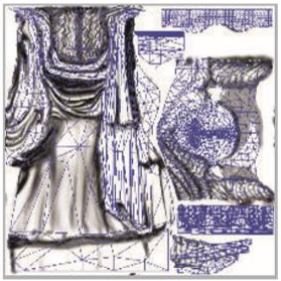
The projector Function

- The spherical projection casts points onto an imaginary sphere centered around some point.
- Cylindrical projection computes the u texture coordinate the same as spherical projection, with the v texture coordinate computed as the distance along the cylinder's axis. Useful for objects with a natural axis.
- The planar projection is like an x-ray beam, projecting in parallel along a direction and applying the texture to all surfaces. Use for applying decals.

The projector Function



How varies texture projections are used on a single model.



Several smaller textures for the statue model, saved in two larger textures. The right figure shows how the triangle mesh is unwrapped and displayed on the texture to aid in its creation.

The Corresponder Function

- converting texture coordinates to texture-space locations.
- one example of it : use the API to select a portion of an existing texture for display for subsequent operations
- Another type: a matrix transformation, enables to translating, rotating, scaling, shearing, or projecting the texture on the surface

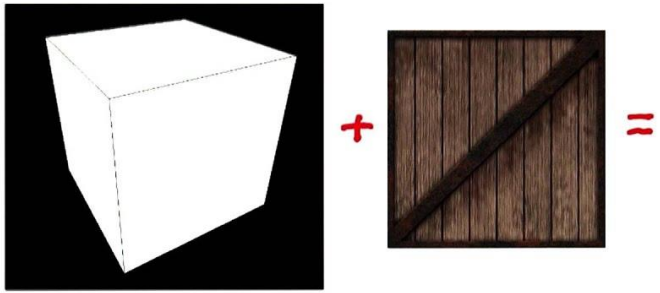
The Corresponder Function

- An image will appear on the surface where (u,v) are in the [0,1] range. But what happens outside of this range?

- A type of corresponder functions determine the behavior.
  - it is called “wrapping mode” in OpenGL, and “texture addressing mode” in DirectX.

The Corresponder Function

- Wrap: useful for having an image of a material repeatedly cover a surface.
- Mirror: provides some continuity along the edges of the texture.
- Clamp: useful for avoiding accidentally taking samples from the opposite edge of a texture when bilinear interpolation happens near a texture's edge
- Border: good for rendering decals onto single-color surfaces.



Play the video
(texturing.mp4)

Image Texturing

- Assume that we have a texture of size 256×256 texels and that we want to use it as a texture on a square.
- As long as the projected square on the screen is roughly the same size as the texture, the texture on the square looks almost like the original image.
- But what happens if the projected square covers ten times as many pixels as the original image contains (called magnification), or if the projected square covers only a small part of the screen (minification)?

it depends on what kind of sampling and filtering methods you decide to use ！！！

Magnification



Texture magnification of a 48×48 image onto 320×320 pixels.

Left: nearest neighbor filtering, where the nearest texel is chosen per pixel. Middle: bilinear filtering using a weighted average of the four nearest texels. Right: cubic filtering using a weighted average of the 5×5 nearest texels.

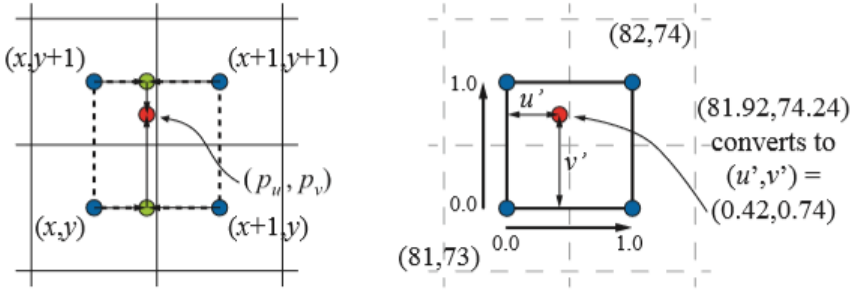
Magnification



Left: the individual texels become apparent because the method requires only one texel to be fetched per pixel, resulting in a blocky appearance.

Middle: blurrier, and much of the jaggedness from using the nearest neighbor method has disappeared

Bilinear interpolation



Bilinear interpolation. The four texels involved are illustrated by the four squares on the left, texel centers in blue. On the right is the coordinate system formed by the centers of the four texels.

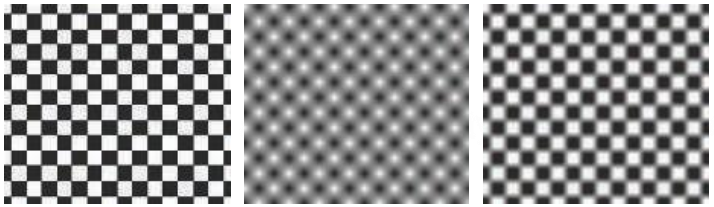
Bilinear interpolation

(u',v'): the location formed by the four texel centers
t(x,y): the texture access function
x,y: the color of the texel is returned

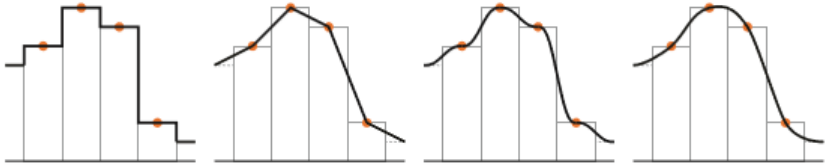
Bilinear interpolation interpolates linearly in two directions.
The bilinearly interpolated color for any location (u',v') can be computed as a two-step process:

- Interpolate t(x,y) and t(x+1,y), t(x,y + 1) and t(x + 1,y + 1) using u'
- Interpolate using v' to get the interpolated color b: →

$$\begin{aligned}\mathbf{b}(p_u, p_v) &= (1-v')\big((1-u')\mathbf{t}(x,y) + u'\mathbf{t}(x+1,y)\big) \\ &\quad + v'\big((1-u')\mathbf{t}(x,y+1) + u'\mathbf{t}(x+1,y+1)\big) \\ &= (1-u')(1-v')\mathbf{t}(x,y) + u'(1-v')\mathbf{t}(x+1,y) \\ &\quad + (1-u')v'\mathbf{t}(x,y+1) + u'v'\mathbf{t}(x+1,y+1).\end{aligned}$$

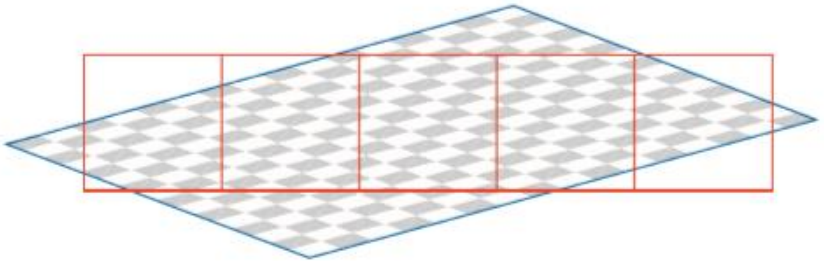


Nearest neighbor, bilinear interpolation, and part way in between by remapping, using the same 2 × 2 checkerboard texture.
Note how nearest neighbor sampling gives slightly different square sizes, since the texture and the image grid do not match perfectly.



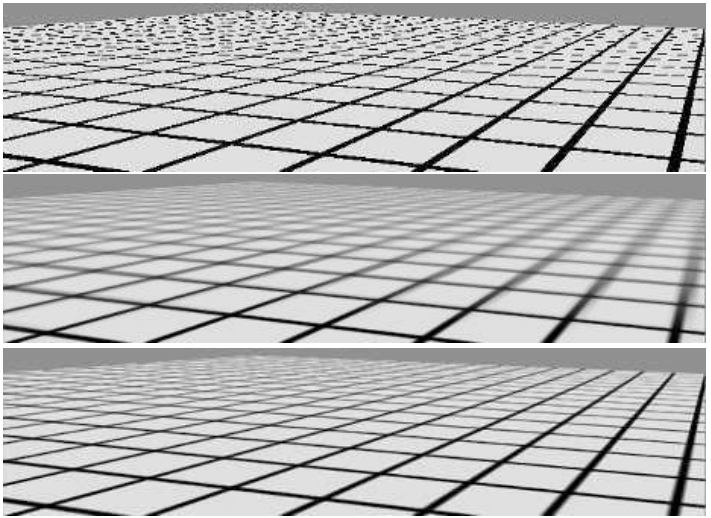
Four different ways to magnify a one-dimensional texture.
The orange circles indicate the centers of the texels as well as the texel values (height).
From left to right: nearest neighbor, linear, using a quintic curve between each pair of neighboring texels, and using cubic interpolation

Minification



When a texture is minimized, several texels may cover a pixel's cell.
A view of a checkerboard-textured square through a row of pixel cells, showing roughly how a number of texels affect each pixel.

Minification



rendered with
nearest neighbor

rendered with
mipmapping

rendered with
summed area tables

Mipmapping

The most popular method of antialiasing for textures is called mipmapping

"Mip" stands for multum in parvo, Latin for "many things in a small place" —a good name for a process in which the original texture is filtered down repeatedly into smaller images

Two important elements in forming high-quality mipmaps are good filtering and gamma correction

Benefit: instead of trying to sum all the texels that affect a pixel individually, precombined sets of texels are accessed and interpolated.
Drawback: overburring

Summed-Area Tabel

- Method to avoid overburring
- An example of *anisotropic filtering* algorithm
- Is able to retrieve texel values effectively in primarily horizontal and vertical directions.
- High quality at a reasonable overall memory cost, can be implemented on modern GPUs

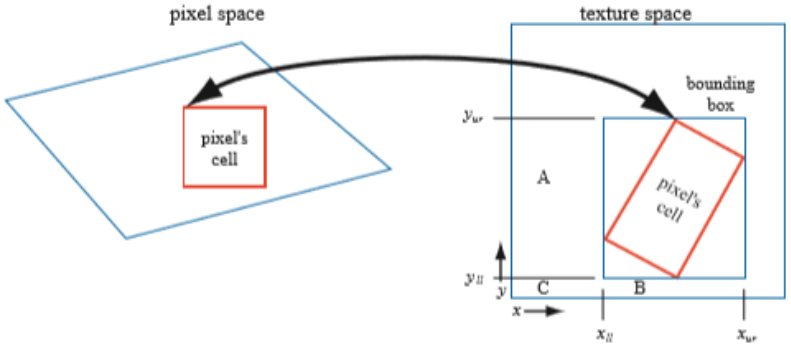
Summed-Area Tabel

- To use it, first creates an array contains more bit of precision for the color stored
- At each location in the array, compute and store the sum of all the corresponding texture's texels in the rectangle formed by this location and the origin
- Then determine the average color using SAT, by the formula:

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

x and y are the texel coordinates of the rectangle and s[x,y] is the summed-area value for that texel

Summed-Area Tabel



The pixel cell is back-projected onto the texture, bound by a rectangle; the four corners of the rectangle are used to access the summed-area table.

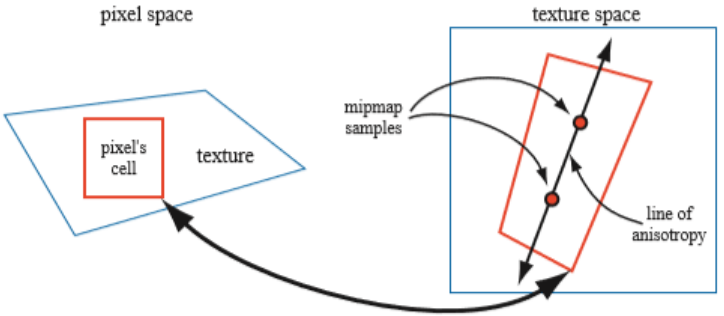
Unconstrained Anisotropic Filtering

For current graphics hardware, the most common method to further improve texture filtering is to reuse existing mipmap hardware

The basic idea is that the pixel cell is back-projected, this quadrilateral (quad) on the texture is then sampled several times, and the samples are combined

Uses several squares to cover the quad. The shorter side of the quad can be used to determine d , and the longer side is used to create a line of anisotropy parallel to the longer side and through the middle of the quad.

Unconstrained Anisotropic Filtering



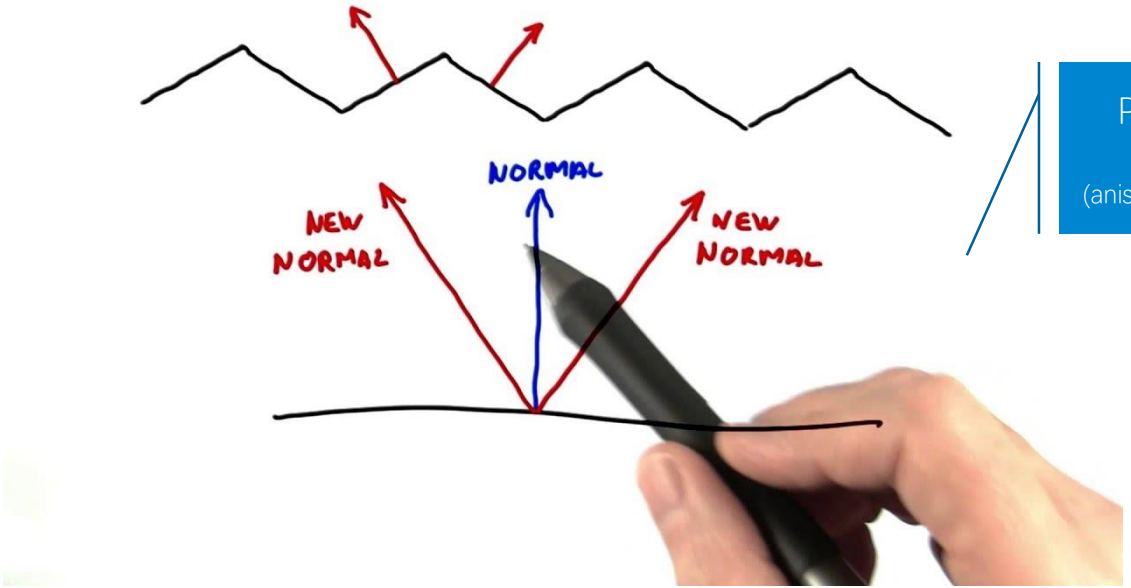
Anisotropic filtering. The back-projection of the pixel cell creates a quadrilateral. A line of anisotropy is formed between the longer sides.

Mipmap versus anisotropic filtering

Left: Trilinear mipmapping. Right: 16 : 1 anisotropic filtering on the right. Toward the horizon, anisotropic filtering provides a sharper result, with minimal aliasing

Online Examples

1. https://threejs.org/examples/#webgl_materials_texture_anisotropy
2. http://www.realtimerendering.com/erich/udacity/exercises/unit3_anisotropy_solution.html



Play the video
(anisotropic.mp4)

Play the video
(anisotropic2.mp4)

Volume Textures

A direct extension of image textures is three-dimensional image data that is accessed by (u,v,w) or (s,t,r) values.

For example, medical imaging data can be generated as a three-dimensional grid; by moving a polygon through this grid, one may view two-dimensional slices of these data.

- Advantages: t he complex process of finding a good 2D parameterization for 3D mesh can be skipped, avoiding the distortion and stem problems
- Disadvatanges: higher storage requirements, more expensive to filter. Inefficient for surface texturing

Textures Representation

Texture atlas: one put several images into a single larger texture.
But may cause problems with mipmapping and repeat modes.

Texture array: an API construction, completely avoids any problems above.

Bindless textures: helps avoid state change cost, makes rendering faster.

Textures Representation

Left: a texture atlas where nine smaller images have been composited into a single large texture.
Right: a more modern approach is to set up the smaller images as an array of textures, which is a concept found in most APIs.

Textures Compression

By having the GPU decode compressed textures on the fly, a texture can require less texture memory and so increase the effective cache size.

A non-compressed texture using 3 bytes per texel at 512*512 resolution would occupy 768 kB. Using texture compression, with a compression ratio of 6 : 1, a 1024*1024 texture would occupy only 512 kB.

Texture compression formats

Name(s)	Storage	Ref colors	Indices	Alpha	Comment
BC1/DXT1	8 B/4 bpt	RGB565×2	2 bpt	–	1 line
BC2/DXT3	16 B/8 bpt	RGB565×2	2 bpt	4 bpt raw	color same as BC1
BC3/DXT5	16 B/8 bpt	RGB565×2	2 bpt	3 bpt interp.	color same as BC1
BC4	8 B/4 bpt	R8×2	3 bpt	–	1 channel
BC5	16 B/8 bpt	RG88×2	2 × 3 bpt	–	2× BC4
BC6H	16 B/8 bpt	see text	see text	–	For HDR; 1–2 lines
BC7	8 B/4 bpt	see text	see text	optional	1–3 lines

All of these compress blocks of 4×4 texels. The storage column show the number of bytes (B) per block and the number of bits per texel (bpt). The notation for the reference colors is first the channels and then the number of bits for each channel. For example, RGB565 means 5 bits for red and blue while the green channel has 6 bits.

BC6H

- BC6H is for high dynamic range (HDR) textures
- each texel initially has 16-bit floating point value per R, G, and B channel.
- This mode uses 16 bytes
  bpt.
- It has one
  another for
  select from a
- Two reference
  encoded for be
  have different accuracy depending on which mode is being used.

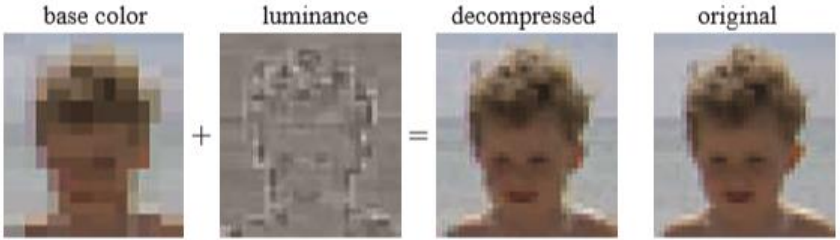
BC7

- In BC7, each
  ween one
  ompression
  C6H, but is a
  ures, while BC6H is for

Note that BC6H and BC7 are called BPTC FLOAT and BPTC, respectively, in OpenGL. These compression techniques can be applied to cube or volume textures.

The main drawback of these compression schemes is that they are lossy. That is, the original image usually cannot be retrieved from the compressed version

For OpenGL ES, another compression algorithm, called Ericsson texture compression (ETC) was chosen for inclusion in the API. This scheme has the same features as S3TC, namely, fast decoding, random access, no indirect lookups, and fixed rate.



ETC encodes the color of a block of pixels and then modifies the luminance per pixel to create the final texel color.

- The effect of using 16 bits per component versus 8 bits during texture compression.
- From left to right: original texture, DXT1 compressed from 8 bits per component, and DXT1 compressed from 16 bits per component with renormalization done in the shader.
- The texture has been rendered with strong lighting in order to more clearly show the effect.

- It is also possible to compress textures in a different color space to speed up texture compression
- RGB⟶YCoCg and inverse transform:

$$\begin{pmatrix} Y \\ C_o \\ C_g \end{pmatrix} = \begin{pmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 0 & -1/2 \\ -1/4 & 1/2 & -1/4 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$G = (Y + C_g), \quad t = (Y - C_g), \quad R = t + C_o, \quad B = t - C_o.$$

- For textures that are dynamically created on the CPU, it may be better to compress the textures on the CPU as well.
- When textures are created through rendering on the GPU, it is usually best to compress the textures on the GPU as well.

Procedural Texturing

- A surface or volume attribute can be:
  - Calculated from a mathematical model
  - Derived in a procedural algorithmic manner
- Procedural Texturing:
  - Does not use intermediate parametric space
  - Often referred to as “procedural shaders”
- Can be used to calculate:
  - A color triplet
  - A normalized set of coordinates
  - A vector direction
  - A scalar value

Properties of Procedural Textures

- Continuous input parameters and continuous output
- No magnification artifacts
- No distortion due to parametric mapping issues
- Map the entire input domain to the output domain

- Due to lack of local control (something that texture images provide), we often combine procedural and image texturing

Noise

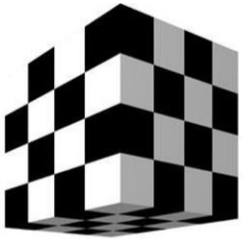
- In nature there are materials and surfaces with irregular patterns, such as a rough wall, a patch of sand, various minerals, stones etc.
- A procedurally generated noise texture should:

– Act as a pseudo-number generator
– Have some controllable properties
– Ensure a consistent output

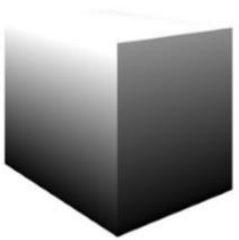
Procedural Noise Properties

- Stateless
- Time-invariant
- Smooth
- Band-limited

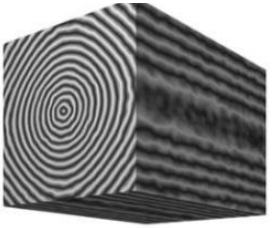
Common Procedurals



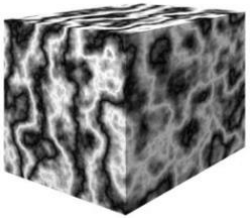
Checker



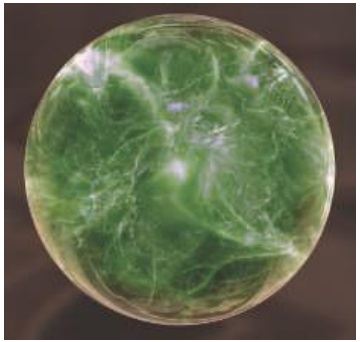
Gradient



Wood



Marble



- Two examples of real-time procedural texturing using a volume texture.
- The marble on the left is a semitransparent volume texture rendered using ray marching.
- On the right, the object is a synthetic image generated with a complex procedural wood shader and composited atop a real-world environment.

Material Mapping

- A common use of a texture is to modify a material property affecting the shading equation
- Real-world objects usually have material properties that vary over their surface.
- To simulate such objects, the pixel shader can read values from textures and use them to modify the material parameters before evaluating the shading equation.
- The parameter that is most often modified by a texture is the surface color.
- Any parameter can be modified by a texture: replacing it, multiplying it, or changing it in some other way.

albedo texture

roughness texture

heightfield texture

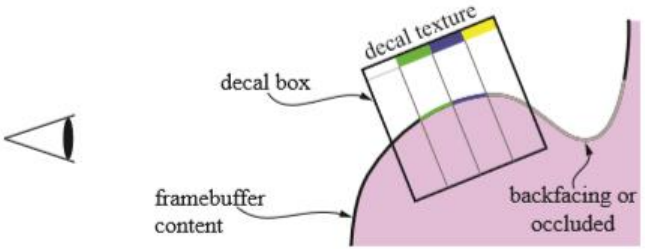
Metallic bricks and mortar. On the right are the textures for surface color, roughness (lighter is rougher), and bump map height (lighter is higher). three different textures are applied to a surface, replacing the constant values

Online Example

https://threejs.org/examples/#webgl_tonemapping

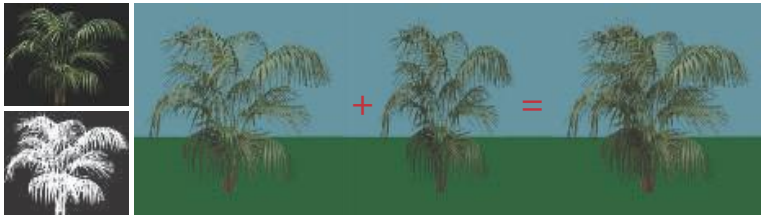
Alpha Mapping

Decaling – one texture related effect



One way to implement decals. The framebuffer is first rendered with a scene, and then a box is rendered and for all points that are inside the box, the decal texture is projected to the framebuffer contents. The leftmost texel is fully transparent so it does not affect the framebuffer. The yellow texel is not visible since it would be projected onto a hidden part of the surface.

Making cutouts by alpha mapping



On the left, the bush texture map and the 1-bit alpha channel map below it.
On the right, the bush rendered on a single rectangle; by adding a second copy of the rectangle rotated 90 degrees, we form an inexpensive three-dimensional bush.

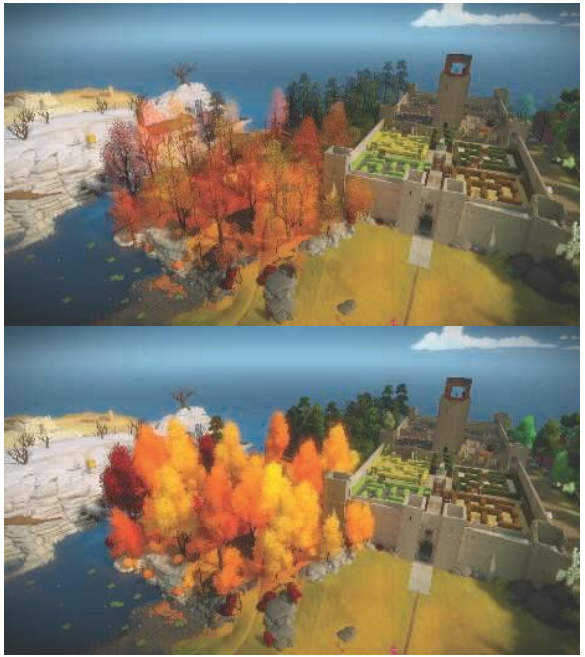
Looking at the “cross-tree” bush from a bit off ground level, then further up, where the illusion breaks down.

To combat this, more cutouts can be added in different ways—slices, branches, layers—to provide a more convincing model.

For mipmap level k, the coverage ck is defined as

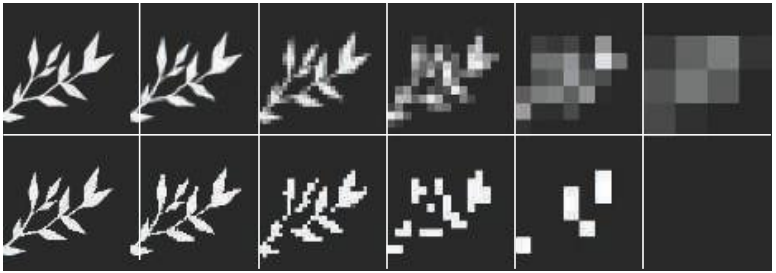
$$c_k = \frac{1}{n_k} \sum_i (\alpha(k,i) > \alpha_t)$$

where n_k is the number of texels in mipmap level k, $\alpha(k,i)$ is the alpha value from mipmap level k at pixel i, and α_t is the user-supplied alpha threshold
Here, we assume that the result of $\alpha(k,i) > \alpha_t$ is 1 if it is true, and 0 otherwise. Note that $k = 0$ indicates the lowest mipmap level, i.e., the original image. For each mipmap level, we then find a new mipmap threshold value α_k, instead of using α_t, such that c_k is equal to c_0 (or as close as possible). This can be done using a binary search. Finally, the alpha values of all texels in mipmap level k are scaled by α_t/α_k.



alpha testing with mipmapping without any correction

alpha testing with alpha values rescaled according to coverage



On the top are the different mipmap levels for a leaf pattern with blending, with the higher levels zoomed for visibility.
On the bottom the mipmap is displayed as it would be treated with an alpha test of 0.5, showing how the object has fewer pixels as it recedes

Different rendering techniques of leaf textures with partial alpha coverage for the edges. From left to right: alpha test, alpha blend, alpha to coverage, and alpha to coverage with sharpened edges.

Bump Mapping

Bump Mapping

Detail on an object can be classified into three scales:

- Macro-features that cover many pixels. When creating a 3D character, the limbs and head are typically modeled at a macroscale.
- Micro-features that are substantially smaller than a pixel, and are encapsulated in the shading model.
- Meso-features that describe everything between these two scales. The wrinkles on a character's face, musculature details, and folds and seams in their clothing, are all mesoscale.
- A family of methods collectively known as bump mapping techniques are commonly used for mesoscale modeling. These adjust the shading parameters at the pixel level

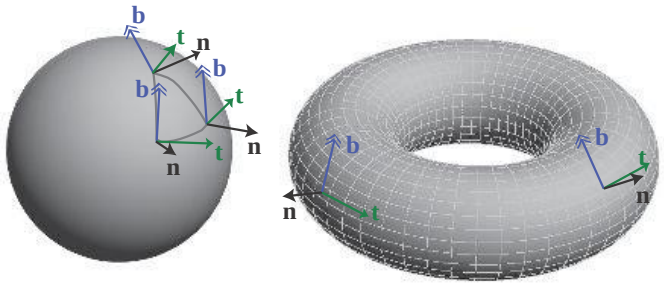
Bump Mapping

Key idea: access a texture to modify the surface normal, instead of using a texture to change a color component in the illumination equation. (The geometric normal remains the same; merely modify the normal used in the lighting equation.)
This matrix, sometimes abbreviated as TBN, transforms a light's direction (for the given vertex) from world space to tangent

These three vectors, normal $\mathbf{n}$, tangent $\mathbf{t}$, and bitangent $\mathbf{b}$, form a basis matrix:

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

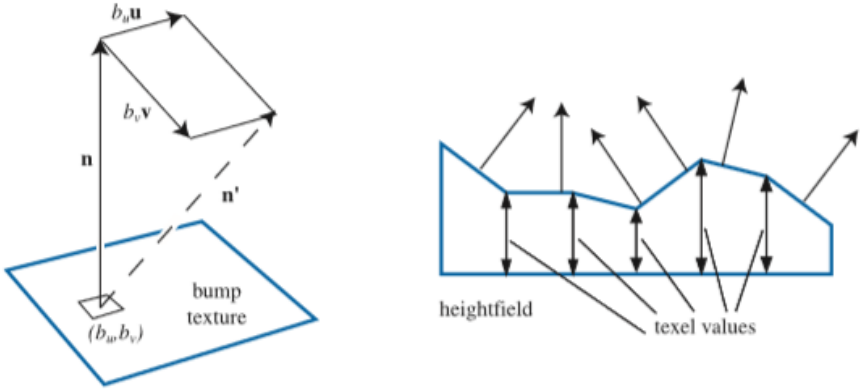
The tangent and bitangent vectors represent the axes of the normal map itself in the object's space



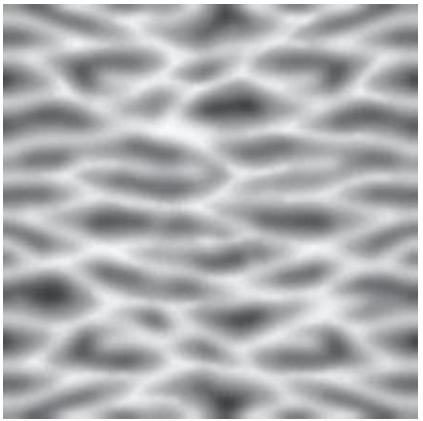
A spherical triangle is shown, with its tangent frame shown at each corner. Shapes like a sphere and torus have a natural tangent-space basis, as the latitude and longitude lines on the torus show.

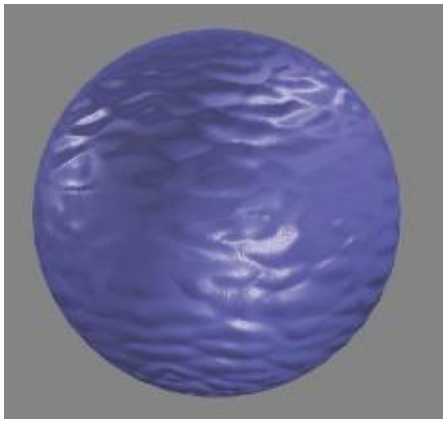
Bump Mapping

- Blinn's Methods:
- Original: stores two signed values, bu and bv, at each texel in a texture. correspond to the amount to vary the normal along the u and v image axes.
- Another way: use a heightfield to modify the surface normal's direction. Each monochrome texture value represents a height, so in the texture, white is a high area and black a low one (or vice versa)



On the left, a normal vector n is modified in the u- and v-directions by the (bu,bv) values taken from the bump texture, giving n′ (which is unnormalized).
On the right, a heightfield and its effect on shading normals is shown. These normals could instead be interpolated between heights for a smoother look.





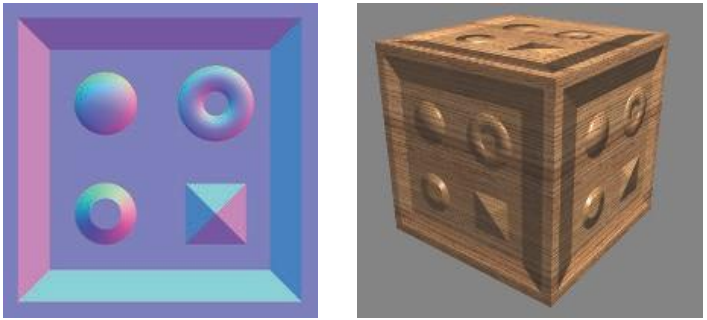
A wavy heightfield bump image and its use on a sphere.

• In bump mapping we implicitly find the diverted normal due to the underlying elevation

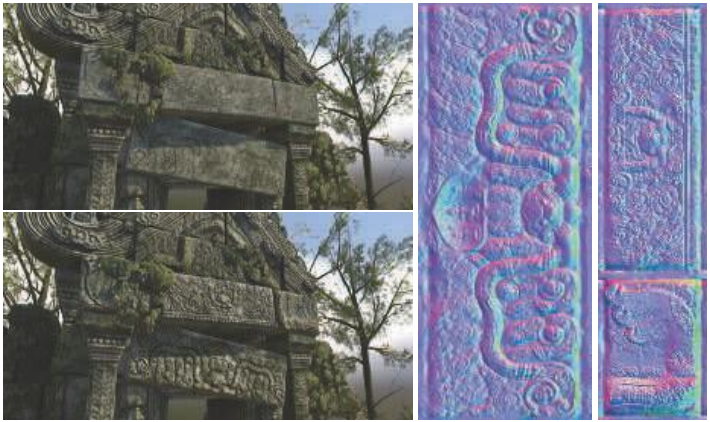
• Normal mapping dispenses with the calculations by directly replacing the local normal with a new normal vector stored in a texture.

• The normal map encodes (x,y,z) mapped to [−1,1], e.g., for an 8-bit texturethe x-axis value 0 represents −1.0 and 255 represents 1.0.

- Calculate and store one tangent vector as additional vertex attribute
- In the vertex shader, calculate and emit the normal and tangent in the same space as the light sources (WCS, ECS), but not in post-projective space
- In the fragment shader:
  - Calculate bitangent via cross product of normal and tangent
  - Fetch new normal from normal map $\mathbf{d}(u, v)$
  - Replace old normal with $\overrightarrow{\mathbf{n}'} = d_x\vec{\mathbf{t}} + d_y\vec{\mathbf{b}} + d_z\vec{\mathbf{n}}$



Bump mapping with a normal map. Each color channel is actually a surface normal coordinate. The red channel is the x deviation; the more red, the more the normal points to the right. Green is the y deviation, and blue is z. At the right is an image produced using the normal map. Note the flattened look on the top of the cube.



- An example of normal map bump mapping used in a game-like scene. Top left: the two normals maps to the right are not applied. Bottom left: normal maps applied. Right: the normal maps.

Parallax Mapping

Why need Parallax Mapping?

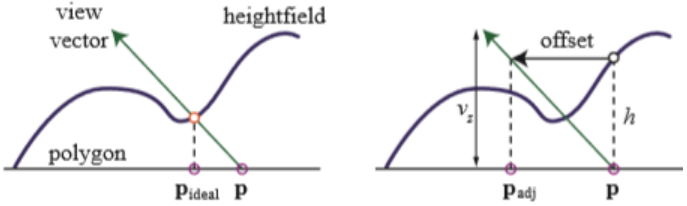
- A problem with bump and normal mapping is that the bumps never shift location with the view angle, nor ever block each other.
- If you look along a real brick wall, for example, at some angle you will not see the mortar between the bricks. A bump map of the wall will never show this type of occlusion, as it merely varies the normal.
- Parallax refers: the positions of objects move relative to one another as the observer moves. As the viewer moves, the bumps should appear to have heights.

- The key idea : take an educated guess of what should be seen in a pixel by examining the height of what was found to be visible.
- For parallax mapping, the bumps are stored in a heightfield texture

Given a texture-coordinate location $\mathbf{p}$, an adjusted heightfield height h, and a normalized view vector $\mathbf{v}$ with a height value v_z and horizontal component $\mathbf{v}_{xy}$, the new parallax-adjusted texture coordinate $\mathbf{p}_{\text{adj}}$ is

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

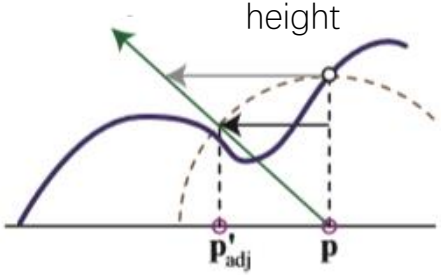
Note that unlike most shading equations, here the space in which the computation is performed matters—the view vector needs to be in tangent space.



On the left is the goal: The actual position on the surface is found from where the view vector pierces the heightfield. Parallax mapping does a first-order approximation by taking the height at the location on the rectangle and using it to find a new location padj

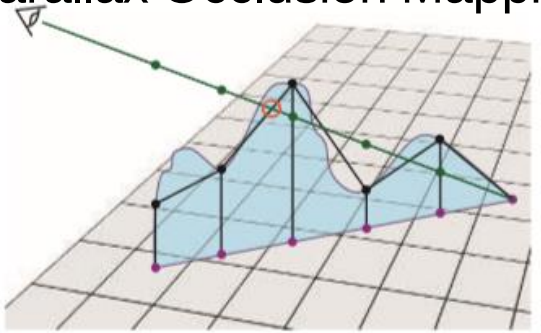
$$\mathbf{p}'_{\text{adj}} = \mathbf{p} + h \cdot \mathbf{v}_{xy}.$$

offset limiting: limit the amount of shifting to never be larger than the retrieved height

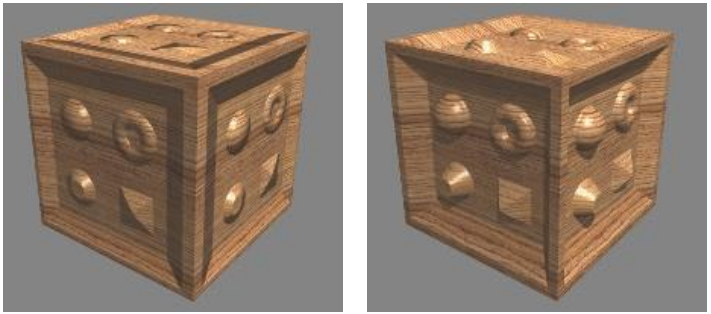


In parallax offset limiting, the offset moves at most the amount of the height away from the original location, shown as a dashed circular arc. The gray offset shows the original result, the black the limited result. On the right is a wall rendered with the technique.

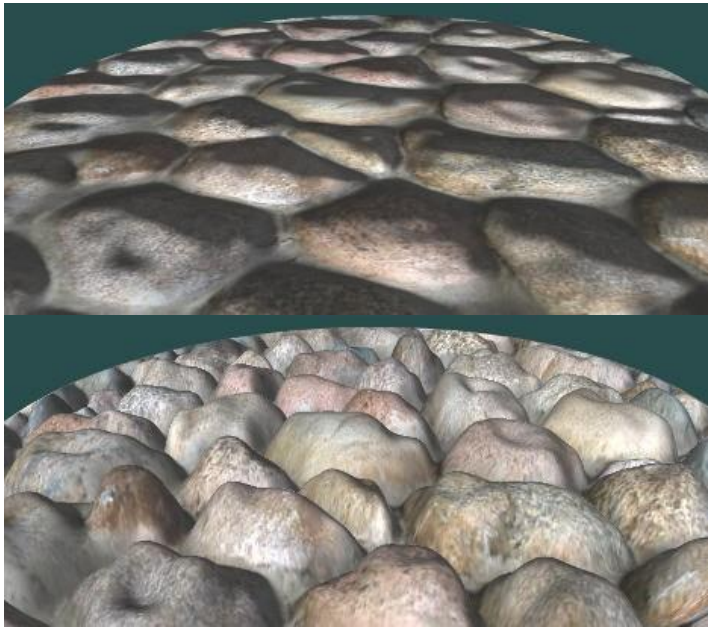
Parallax Occlusion Mapping



The green eye ray is projected onto the surface plane, which is sampled at regular intervals (the violet dots) and the heights are retrieved. The algorithm finds the first intersection of the eye ray with the black line segments approximating the curved height field.



Parallax mapping without ray marching (left) compared to with ray marching (right). On the top of the cube there is flattening when ray marching is not used. With ray marching, selfshadowing effects are also generated



Normal mapping and relief mapping. No self-occlusion occurs with normal mapping. Relief mapping has problems with silhouettes for repeating textures, as the rectangle is more of a view into the heightfield than a true boundary definition

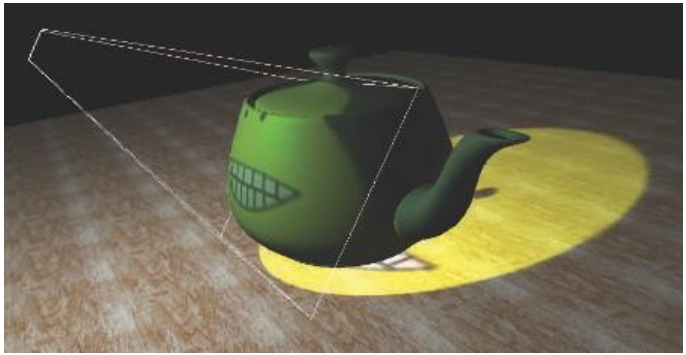
Parallax occlusion mapping, a.k.a. relief mapping, used on a path to make the stones look more realistic. The ground is actually a simple set of triangles with a heightfield applied.

Texture Lights

- Textures can also be used to add visual richness to light sources and allow for complex intensity distribution or spotlight functions.

- For lights that have all their illumination limited to a cone or frustum, projective textures can be used to modulate the light intensity.

- For lights that are not limited to a frustum but illuminate in all directions, a cube map can be used to modulate the intensity, instead of a two-dimensional projective texture.



Projective textured light. The texture is projected onto the teapot and ground plane and used to modulate the light's contribution within the projection frustum (it is set to 0 outside the frustum).

Further Reading and Resources

- The book Advanced Graphics Programming Using OpenGL has extensive coverage of various visualization techniques using texturing algorithms
- For extensive coverage of three-dimensional procedural textures, see Texturing and Modeling: A Procedural Approach .
- The book Advanced Game Development with Programmable Graphics Hardware [1850] has many details about implementing parallax occlusion mapping techniques.
- A website: Shadertoy , There are many worthwhile and fascinating procedural texturing functions on display