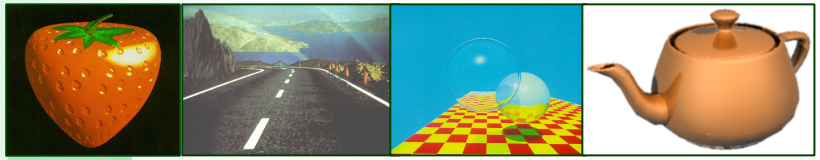


Cap. 9
Texturas

Engenharia Informática (5385) - 2º ano, 2º semestre

Sumário

- Objectivos
- Noção de textura
- Motivação
- Mapeamento de textura, padrões de textura e texels
- Mapeamento de texturas em polígonos, interpolação de texturas
- Rasterização: modos de aplicação de texturas
- Mapeamento de texturas em objectos geométricos
  - mapeamento planar
  - mapeamento cilíndrico
  - mapeamento esférico
  - mapeamento paralepipédico
- Modos de revestimento (wrapping)
- ...

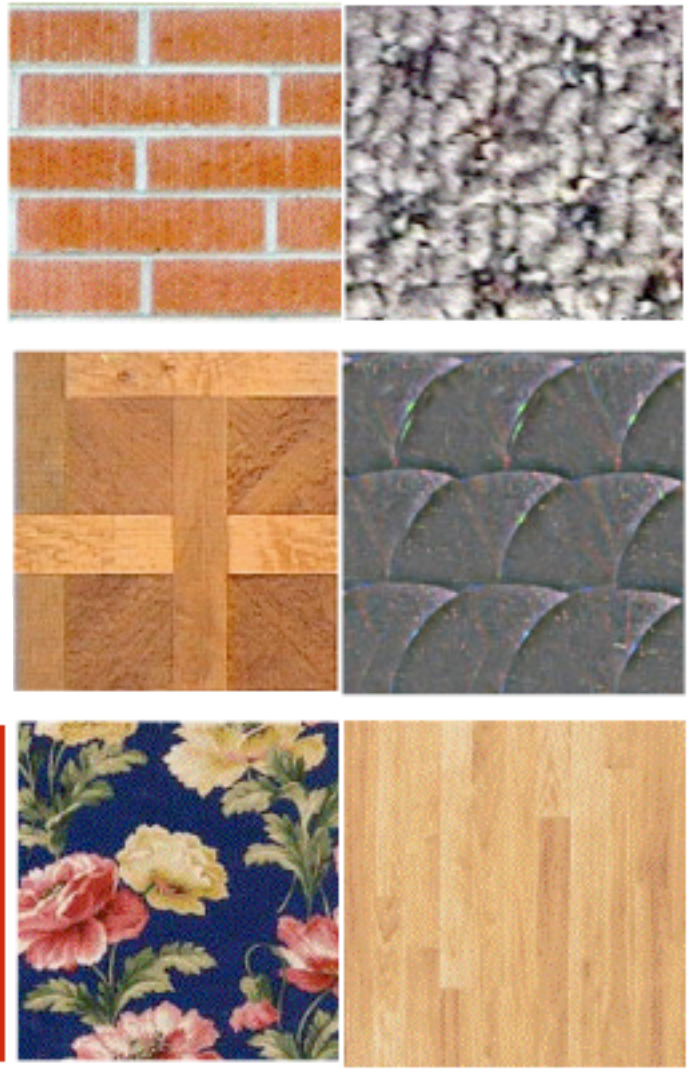
Objectivos

- Introduzir métodos de mapeamento
  - Mapeamento de textura (texture mapping)
  - Mapeamento ambiental (environmental mapping)
  - Mapeamento rugoso (bump mapping)
  - Mapeamento de luz (light mapping)

- Considera-se duas estratégias básicas:
  - Especificação manual de coordenadas
  - Mapeamento automático em duas fases

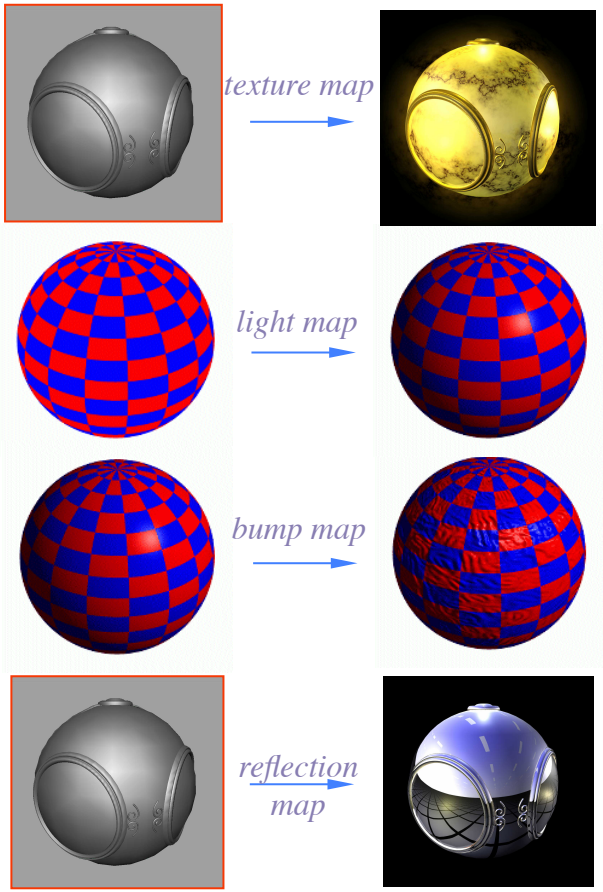
Noção de Textura

- Uma **textura** é uma imagem com componentes RGB e alpha...
- **Mapeamento de textura**: um método para criar complexidade numa imagem sem o ónus de construir grandes modelos geométricos



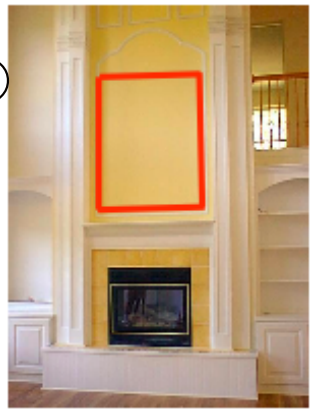
Motivação:
(1) adicionar realismo

- Objectos renderizados com o modelo de reflexão de Phong e com a coloração interpolada de Gouraud/Phong parecem ter uma aparência um pouco 'plástica'
- Os efeitos de textura podem ser adicionados para dar uma aparência mais realística à superfície
  - Mapeamento de textura (texture mapping)
    - Utiliza um padrão que é colocado na superfície dum objecto.
  - Mapeamento de luz (light maps)
    - Light maps combinam textura e luz através de um processo de modulação
  - Mapeamento de rugosidade (bump mapping)
    - A superfície suave é distorcida para obter o efeito rugoso na superfície
  - Mapeamento ambiental
    - É mapeamento de reflexão que permite resultados semelhantes ao ray tracing

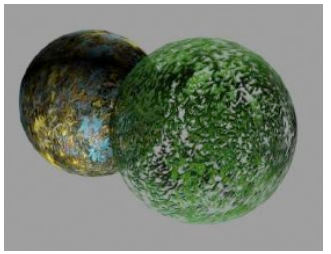


Motivação: (2) adicionar detalhe à superfície

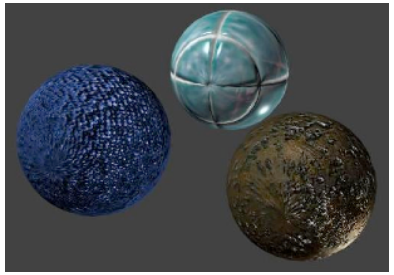
- A solução mais óbvia não é a melhor
  - Dividir uma cena em polígonos cada vez mais pequenos aumenta o detalhe
  - Mas é muito difícil de modelar e muito oneroso de renderizar em termos de tempo
- A solução preferida é usar mapeamento de textura
  - tipicamente uma imagem 2D decalcada em objectos
- *Exemplos*:
  - Modelo de t-shirt com logo ①
    - não há necessidade de modelar as letras nem nenhum processament de triângulos
    - usa-se um polígono de dimensões grandes
    - colora-se o polígono com uma foto
  - Iluminação súbtil da parede ②
    - não há necessidade de calculá-la em cada frame
    - não há necessidade de modelá-la com vários triângulos de cor consta
    - decalca-se a foto num polígono grande
  - Também funciona em superfícies não complanares ③
    - subdivide surface into planar patches
    - assign photograph subregions to each individual patch
  - Exemplos de modulação de cor, rugosidade, brilho, e transparência ④ com a mesma geometria esférica

② 

① 



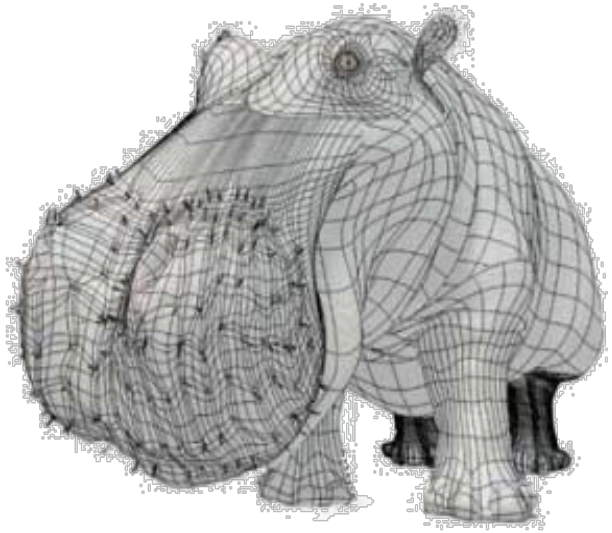
③ 



④

Texturas: em que ponto as coisas começam a parecer reais?

- Superfícies de objectos do mundo “selvagem” são muito complexas
- Não podem modelar as mais finas variações
- Precisamos de encontrar formas de adicionar **detalhe à superfície**. Como?



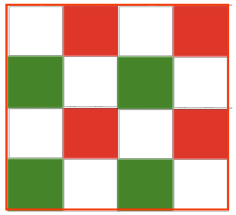
modelo geométrico

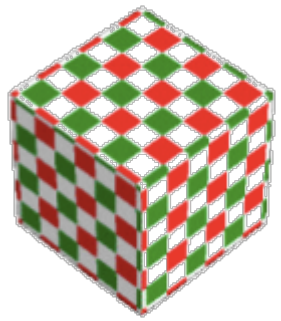
modelo geométrico
+
coloração

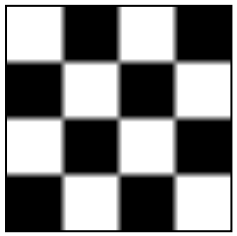
modelo geométrico
+
coloração
+
texturas

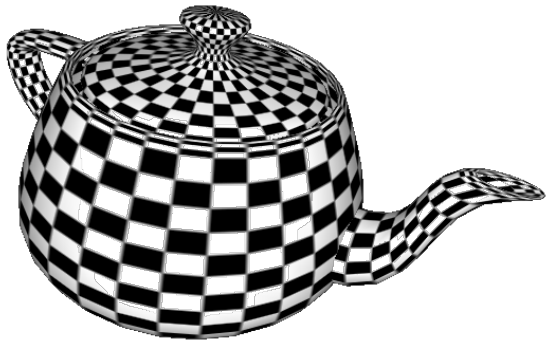
Mapeamento de textura, padrão de textura e texels

- Desenvolvido por Catmull (1974), Blinn e Newell (1976), e outros investigadores.
- **Mapeamento de textura**: adiciona detalhe à superfície através de mapeamento de padrões de texturas sobre a superfície.
- Padrão é repetido. Por exemplo, o **padrão de textura** para o cubo ao lado é o seguinte:









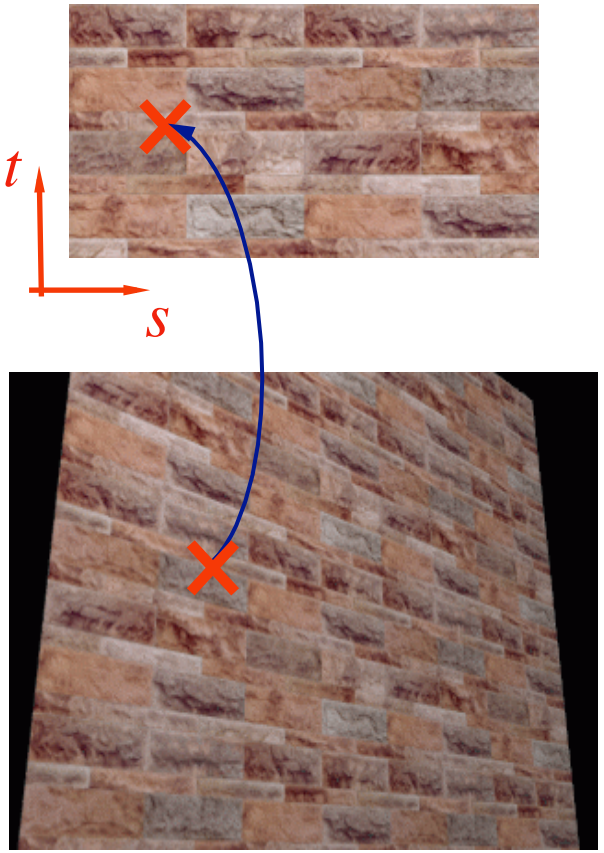
- Texel: abreviatura para “texture element”.
- Um **texel** é um pixel da textura. Por exemplo, uma textura de resolução 128x128 tem 128x128 texels. No ecrã, isto pode resultar em mais ou menos pixels, o que dependerá da distância a que o objecto se encontra na cena, ou seja, de como a textura é escalada para a superfície.

Técnicas de Mapeamento

- Mapeamento de Texturas
- Mapeamento Ambiental
- Mapeamento de Rugosidade
- Mapeamento de Luz

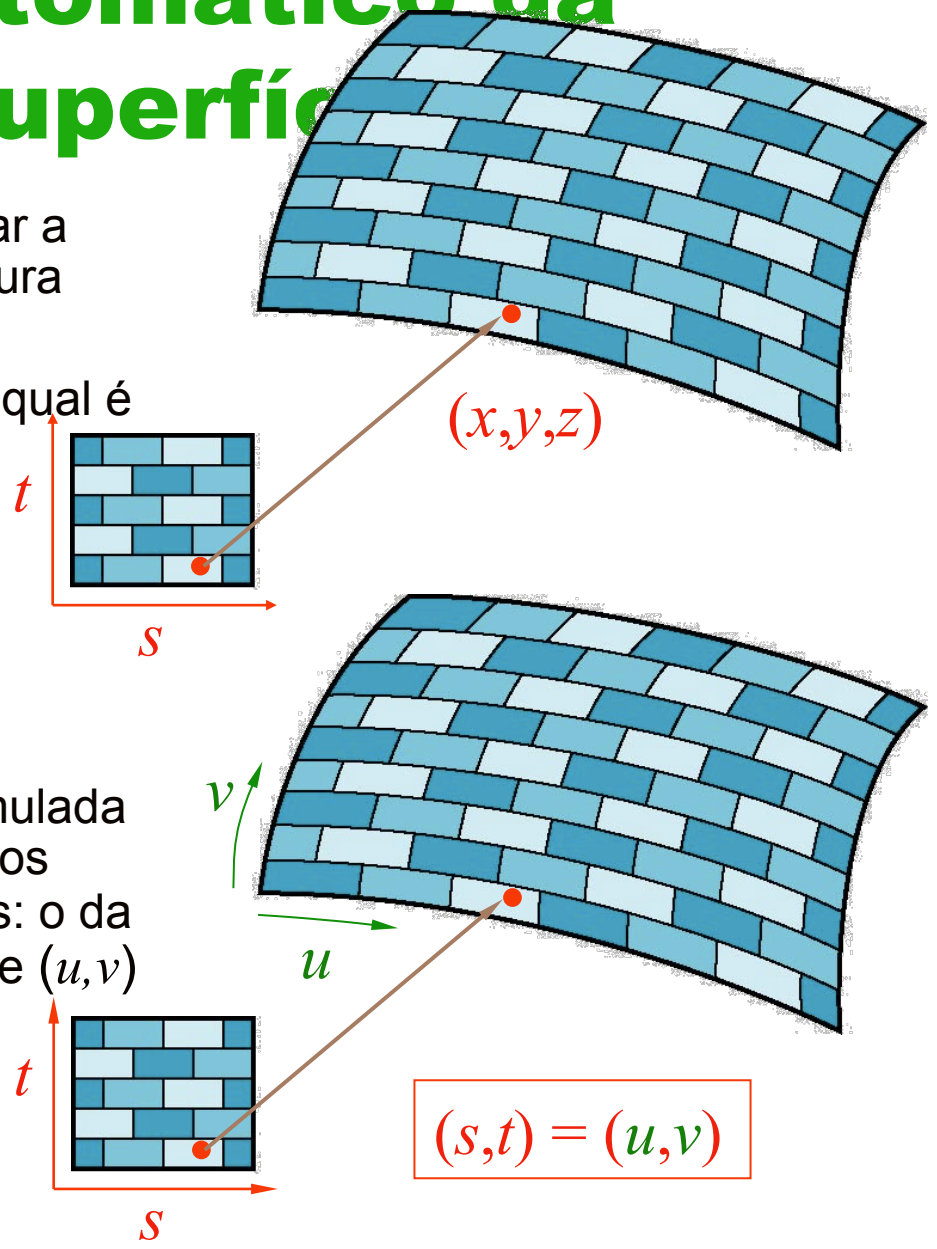
Mapeamento (manual) de Textura

- Questão: Que ponto da textura se usa para um dado ponto da superfície?
- A textura é simplesmente uma imagem com um sistema de coordenadas 2D (s,t)
  - Parameteriza-se pontos da textura com coordenadas: (s,t)
- Define-se o mapeamento de (x,y,z) no domínio da cena para o domínio da textura (s,t)
  - Para determinar a cor em (x,y,z) usa-se o correspondente (s,t) para determinar a cor da textura nesse ponto
- Com polígonos:
  - Especifica-se (s,t) nos vértices
  - Interpola-se (s,t) para os pontos intermédios entre os vértices e no interior do polígono



Mapeamento automático da textura para a superfí

- O problema básico está em encontrar a função (mapping) que mapeia a textura para a superfície
- Dada uma posição (s,t) da textura, qual é a posição (x,y,z) na superfície?
- Este problema requer 3 funções paramétricas:
  - $x = \mathbf{X}(s, t)$
  - $y = \mathbf{Y}(s, t)$
  - $z = \mathbf{Z}(s, t)$
- No caso de a superfície já estar formulada parametricamente, então temos outros sistemas de coordenadas envolvidos: o da imagem (s,t) e o da própria superfície (u,v)

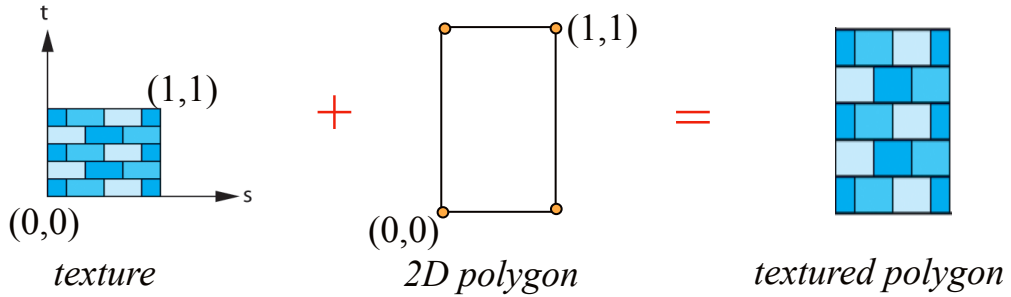


Como se define as coordenadas paramétricas (u,v)?

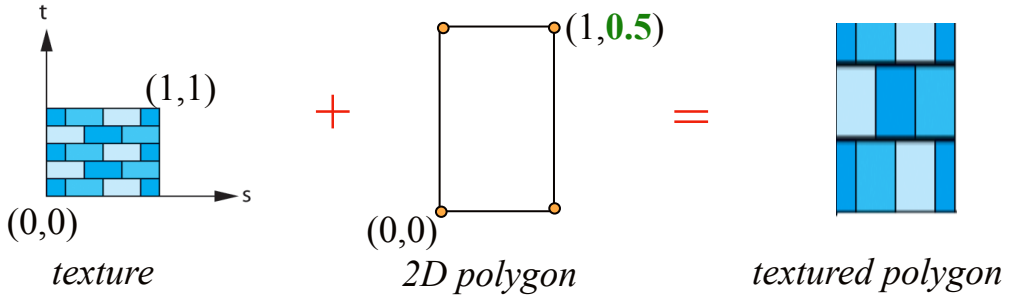
- Manualmente:
  - Nós atribuimos as coordenadas da textura a cada vértice da superfície
- Automaticamente:
  - Usa-se um algoritmo que atribui as coordenadas da textura

Especificação manual de coordenadas

- Podemos especificar manualmente as coordenadas da textura em cada vértice



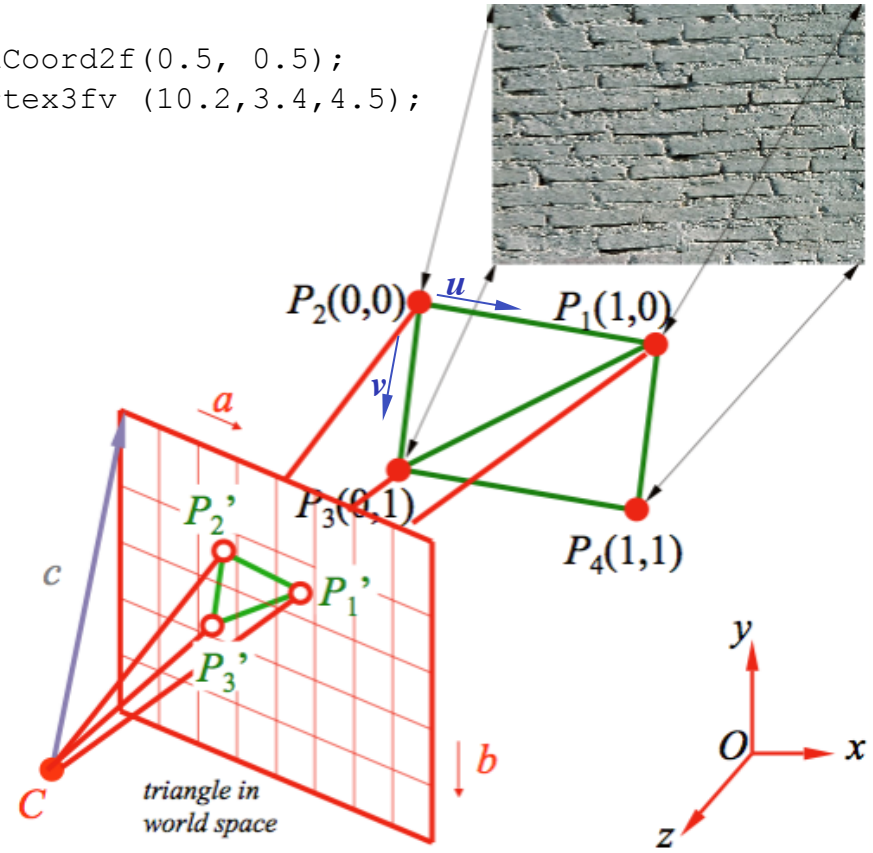
- Podemos escolher coordenadas de textura alternativas



Mapeamento de texturas em polígonos

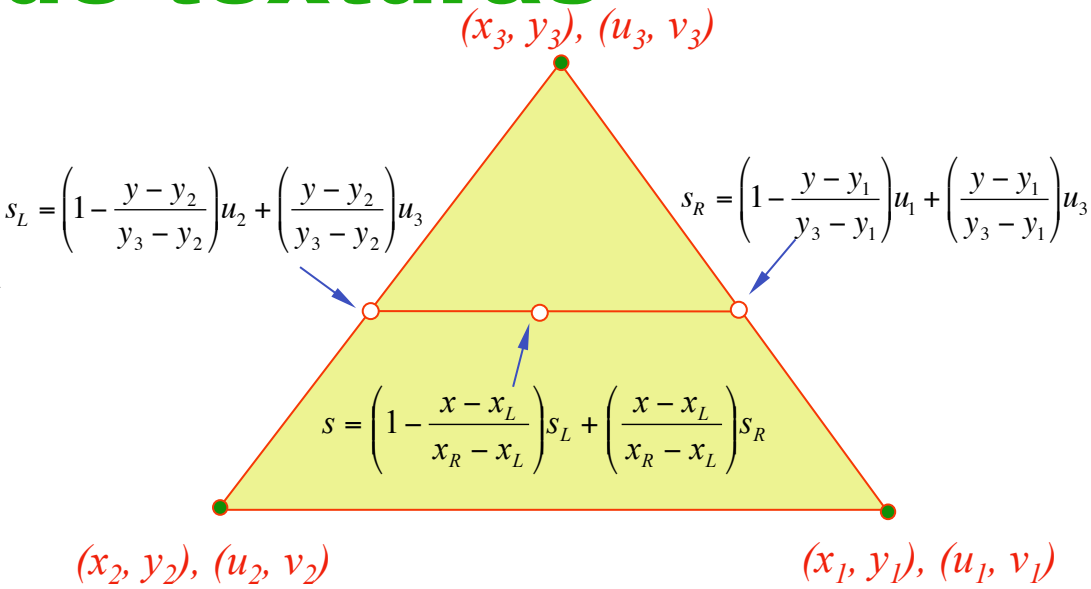
- Especificamos explicitamente as coordenadas (*u*,*v*) nos vértices do polígono
- Ou seja, afixamos a textura nos vértices
- Faz-se então a **interpolação** dentro do triângulo na altura da rasterização (scan conversion) no domínio de ecrã.

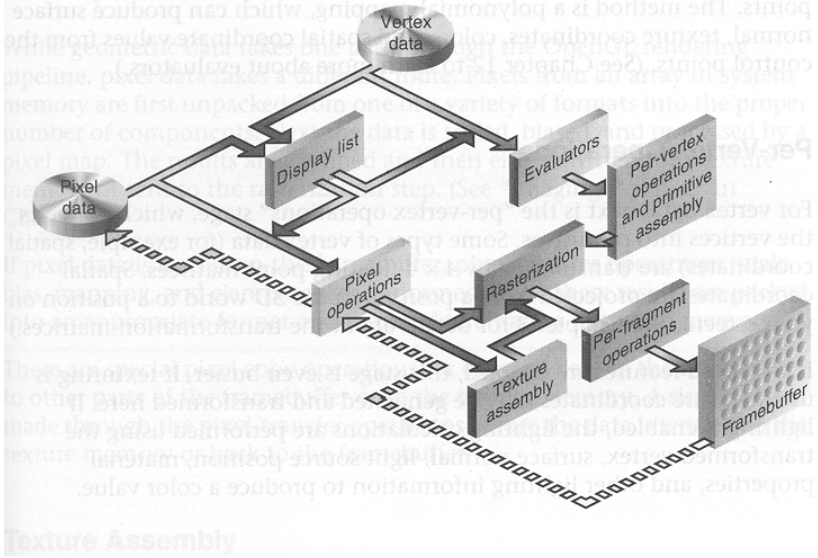
```
…
glTexCoord2f(0.5, 0.5);
glVertex3fv (10.2,3.4,4.5);
…
```



Interpolação de texturas

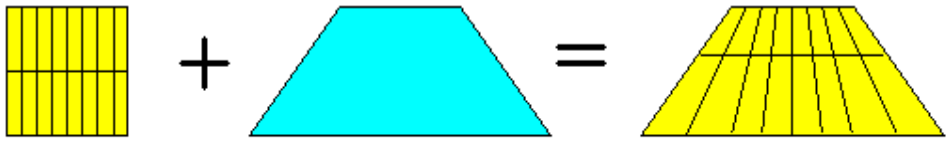
- A interpolação é feita durante a fase de *scan conversion*, à semelhança do que é feito na coloração de Gouraud
- Mas e vez de interpolar os valores RGB, obtemos os valores (*u*,*v*) que permitem aceder à correspondente cor na textura.
- Isto é, o mapeamento da textura é feito no domínio canónico do ecrã quando o polígono é rasterizado
- No entanto, quando se descreve uma cena, assume-se muitas vezes que o mapeamento é feito no domínio da cena



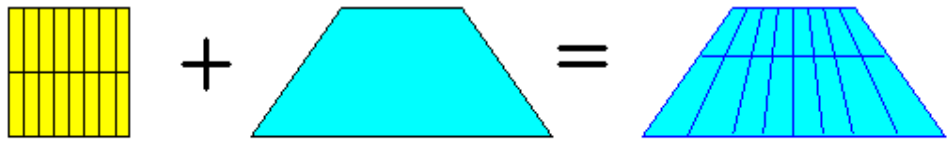


Rasterização: modos de aplicação de texturas

- Após saber os valores de uma textura, podemos usá-los para modificar um ou mais atributos de um polígono/ superfície através das chamadas funções de mistura (***combine functions*** ou ***texture blending operations***):
  - **replace:** substitui cor da superfície pela cor da textura
  - **decal**: substitui cor da superfície pela cor da textura, misturando então a cor com o valor alpha da textura, ainda que a componente alpha não seja modificado no framebuffer
  - **modulate**: multiplica a cor da superfície pela cor da textura (shaded + textured surface). Precisa-se disto para a multi-texturização (i.e., lightmaps).
  - **blend**: semelhante à modulação mas com mistura alpha



`operação REPLACE`



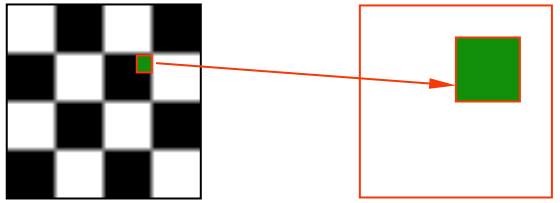
`operação MODULATE`

Questões relevantes em mapeamento de texturas

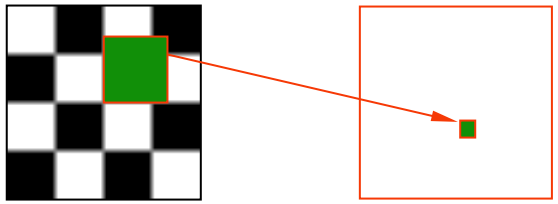
- O que acontece quando se faz zoom aumentativo e zoom diminuitivo?
- Como geramos as coordenadas de texturas?
- O que acontece quando usamos coordenadas de textura menores do que 0 e maiores do que 1?
- Será que as aplicações (maps) de textura só servem para por cor nos objectos?

Aplicação de uma textura numa superfície

- A aplicação de uma textura numa superfície tem lugar durante a renderização.
- É semelhante ao método de coloração de Gouraud:
  - Triângulo rasterizado
  - Cada pixel do triângulo é mapeado de volta à textura
  - Usa-se valores conhecidos nos vértices para interpolar sobre a textura
- Cada pixel está associado com uma pequena região da superfície e a uma pequena área da textura.
- Há 3 possibilidades de associação:
  1. um texel a um pixel (raramente)
  2. magnificação: um texel a muitos pixels
  3. minificação: muitos texels a um pixel



Magnificação

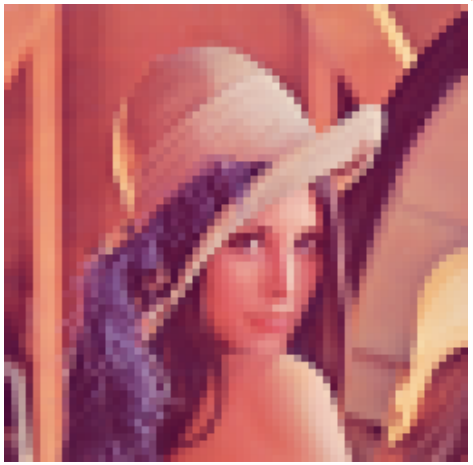


Minificação

Aplicação de uma textura numa superfície (cont.): zoom in: filtro de magnificação

Magnificação

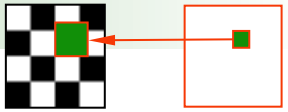
- Um pixel é mapeado para uma pequena porção de *um* texel
- O resultado é que muitos pixels são mapeados para o mesmo texel
- Sem um método de filtragem surge o fenómeno de *aliasing*
- Filtro de magnificação: suaviza a transição entre pixels



muitos pixels correspondem a um texel
→ “blockiness” / jaggies / aliasing

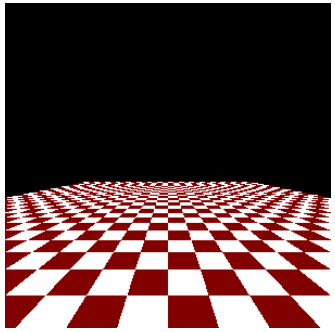
solução: aplicar média
(magnification filter)

Aplicação de uma textura numa superfície (cont.): zoom out: filtro de minificação

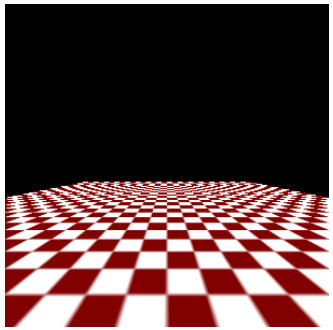


Minificação

- Um pixel é mapeado para muitos texels
- É comum em perspectiva (foreshortening)

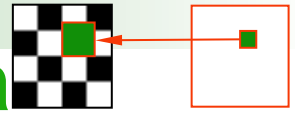


Perspectiva (foreshortening) e mapeamento pobre de textura provoca a deformação visual do pavimento

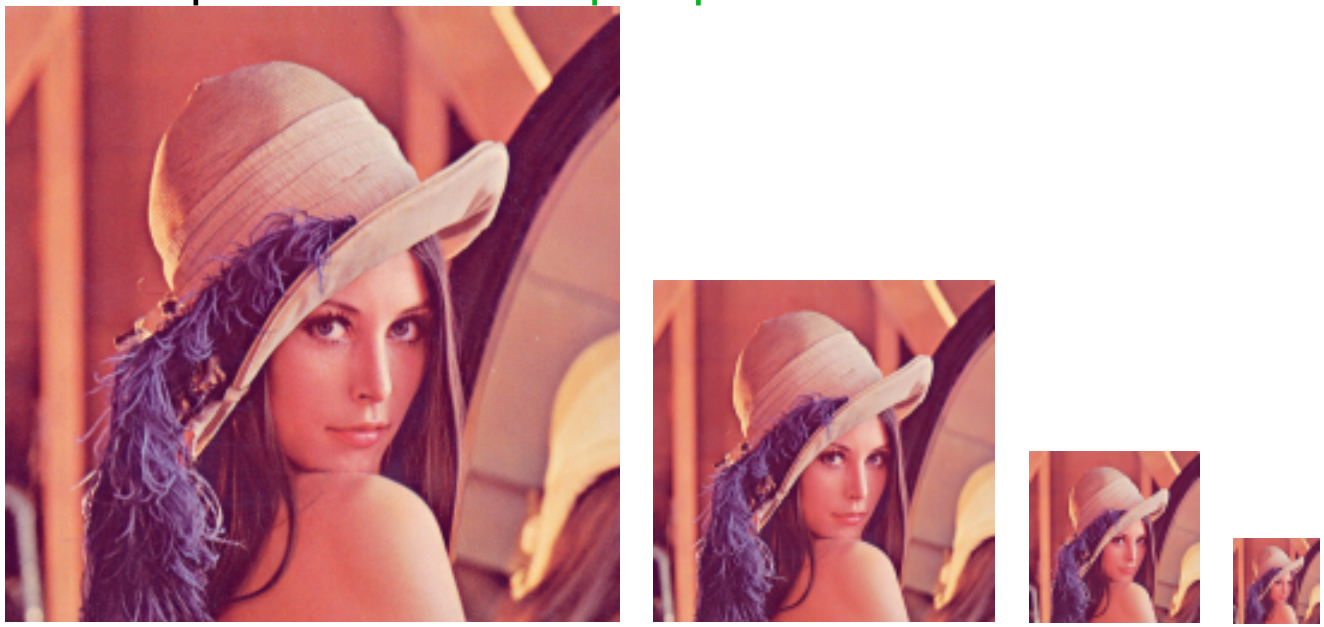


Mipmaps melhoram o mapeamento, restaurando a forma do pavimento

Aplicação de uma textura numa superfície (cont.): melhor filtro de minificação: Mipmaps



- "mip" significa *multum in parvo*, ou "muitas coisas num pequeno lugar"
- Ideia básica: criar muitas texturas de tamanho decrescente e usar uma delas quando for apropriado
- Texturas pré-filtradas = mipmaps



Aplicação de uma textura numa superfície (cont.): Mipmaps: otimização de armazenamento

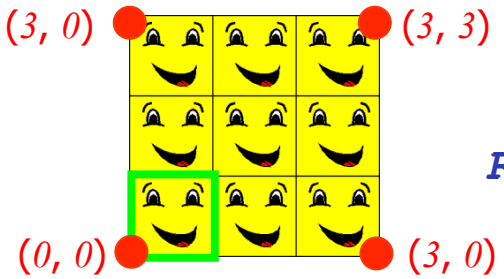
- É obrigatório fornecer todos os tamanhos da textura em potências de 2 em relação à original em 1x1

Sumário: filtragem

- Zoom-in requer a utilizaçao dum filtro de magnificação
- Zoom-out requer a utilizaçao dum filtro de minificação
- Filtros mais avançados requerem mais tempo/computação, mas produzem melhores resultados
- Mipmapping é um filtro de minificação avançado
- Precaução: mipmapping sem pré-definir mipmaps desligará (turn off) a texturização (veja-se *Filtering in OpenGL*)

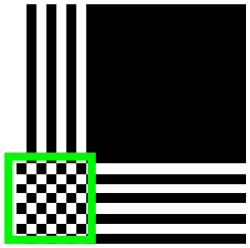
Wrapping Modes

- Can assign texture coords outside of [0,1] and have them either *clamp* or *repeat* the texture map
- **Wrapping modes:**
  - **repeat**: start entire texture over
    - Repeat Issue: making the texture borders match-up
  - **mirror**: flip copy of texture in each direction
    - get continuity of pattern
  - **clamp to edge**: extend texture edge pixels
  - **clamp to border**: surround with border color

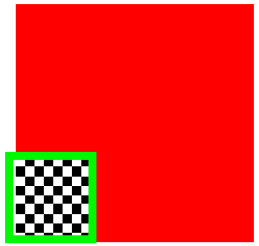


REPEAT

MIRROR



CLAMP TO EDGE



CLAMP TO BORDER

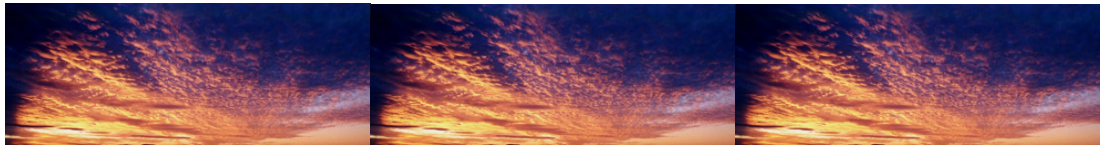
courtesy of Microsoft

Wrapping modes:
Repetitive texture tiling

- A texture can also be repeatedly tiled across the surface by repeating the (s,t) parameterization over the surface

- But, best results are obtained when the texture is seamlessly tilable
  - This means that the right side of the texture joins seamlessly with the left side (same with the bottom and top)

- Seams will appear for most textures when tiled:



- But, we can edit or re-synthesize textures to be seamlessly repeatable (*this is another topic onto itself*):

Texturing in OpenGL

- Procedure
- Example

Texturing in OpenGL

- Procedure: Texture Mapping
  - uploading of the texture to the video memory
  - set up texture parameters
  - enable texturing
  - the application of the texture onto geometry

Texturing in OpenGL: main steps

1. Create texture and load with
   - `glTexImage()`
   - Three methods:
     - read in an image in a jpg, bmp, ... File
     - generate the texture yourself within application
     - copy image from color buffer
2. define texture parameters as to how texture is applied
   - `glTexParameter*()`
   - wrapping, filtering, etc
3. enable texturing
   - `glEnable(GL_TEXTURE_*D)`
4. assign texture coordinates to vertices
   - the mapping function is left up to you
   - `glTexCoord*(s,t);`
   - `glVertex*(x,y,z);`

Step 1: Specifying Texturing Image

- Define a texture image as an array of *texels* (texture elements) in CPU memory:

```
Glubyte myTexture[width][height][3];
```

- Each RGB value is specified to be an unsigned byte, between 0 and 255

- For example, a blue color would be (0, 0, 255)

Warning: *this [3] seems to be missing in the textbook*

Step 1: Defining an Image as a Texture

- Call `glTexImage2D`. It uploads the texture to the video memory where it will be ready for us to use in our programs.

- `void glTexImage2D(target,level,components,w,h,border,format,type,texture);`
  Parameters:
  - `target` : type of texture, e.g. `GL_TEXTURE_2D`
  - `level` : used for mipmapping = 0 (discussed later)
  - `components` : elements per texel (for RGB)
  - `w, h` : width and height of texture in pixels
  - `border` : used for smoothing = 0 (don't worry about this)
  - `format` : `texel` format e.g. `GL_RGB`
  - `type` : rgb component format e.g. `GL_UNSIGNED_BYTE`
  - `texture` : pointer to the texture array

- Example, set the current texture:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, texture);
```

Step 1 (*alternative*): Generating a Random Texture

- Here is a function for a random texture:

```
GLubyte texture[64][64][3];
int u, v;

for(u=0; u<64; u++)
{
     for(v=0; v<64; v++)
     {
       texture[u][v][0] = (GLubyte)(255 * rand()/RAND_MAX);
       texture[u][v][1] = (GLubyte)(255 * rand()/RAND_MAX);
       texture[u][v][2] = (GLubyte)(255 * rand()/RAND_MAX);
     }
}
```

Step 2: Specifying Texture Parameters

- OpenGL has a variety of parameters that determine how textures are applied:
  - Wrapping parameters determine what happens if s and t are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
- The `glTexParameter()` function is a crucial part of OpenGL texture mapping, this function determines the behavior and appearance of textures when they are rendered.
- Take note that each texture uploaded can have its own separate properties, texture properties are not global.

Step 2:
glTexParameter()

Target	Specifies the target texture
GL_TEXTURE_1D	One dimensional texturing.
GL_TEXTURE_2D	Two dimensional texturing.

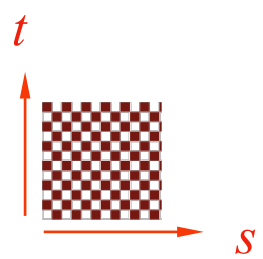
Texture Parameter	Accepted values	Description
GL_TEXTURE_MIN_FILTER	GL_NEAREST , GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR and GL_LINEAR_MIPMAP_LINEAR	The texture minification function is used when a single screen pixel maps to more than one texel, this means the texture must be shrunk in size. Default setting is GL_NEAREST_MIPMAP_LINEAR.
GL_TEXTURE_MAG_FILTER	GL_NEAREST or GL_LINEAR	The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texel, this means the texture must be magnified. Default setting is GL_LINEAR.
GL_TEXTURE_WRAP_S	GL_CLAMP or GL_REPEAT	Sets the wrap parameter for the s texture coordinate. Can be set to either GL_CLAMP or GL_REPEAT. Default setting is GL_REPEAT.
GL_TEXTURE_WRAP_T	GL_CLAMP or GL_REPEAT	Sets the wrap parameter for the t texture coordinate. Can be set to either GL_CLAMP or GL_REPEAT. Default setting is GL_REPEAT.
GL_TEXTURE_BORDER_COLOR	Any four values in the [0, 1] range	Sets the border color for the texture, if border is present. Default setting is (0, 0, 0, 0).
GL_TEXTURE_PRIORITY	[0, 1]	Specifies the residence priority of the texture, use to prevent OpenGL from swapping textures out of video memory. Can be set to values in the [0, 1] range. See *glPrioritizeTextures()* for more information or this article on Gamasutra.

Step 2:
glTexParameter()

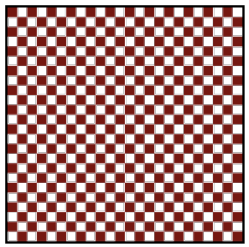
Parameter Value	Description
GL_CLAMP	Clamps the texture coordinate in the [0,1] range.
GL_REPEAT	Ignores the integer portion of the texture coordinate, only the fractional part is used, which creates a repeating pattern. A texture coordinate of 3.0 would cause the texture to tile 3 times when rendered .
GL_NEAREST	Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured. Use this parameter if you would like your texture to appear sharp when rendered .
GL_LINEAR	Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of GL_TEXTURE_WRAP_S and GL_TEXTURE_WRAP_T, and on the exact mapping. Use this parameter if you would like your texture to appear blurred when rendered .
GL_NEAREST_MIPMAP_NEAREST	Chooses the mipmap that most closely matches the size of the pixel being textured and uses the GL_NEAREST criterion (the texture element nearest to the center of the pixel) to produce a texture value.
GL_LINEAR_MIPMAP_NEAREST	Chooses the mipmap that most closely matches the size of the pixel being textured and uses the GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value.
GL_NEAREST_MIPMAP_LINEAR	Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the GL_NEAREST criterion (the texture element nearest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.
GL_LINEAR_MIPMAP_LINEAR	Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

Step 2: Wrapping Modes

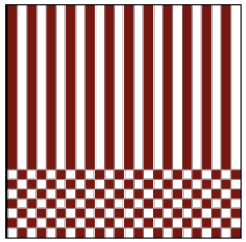
- Clamping : if $s,t > 1$ use color at 1, if $s,t < 0$ use color at 0
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);`
- Repeating : use s,t modulo 1
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);`



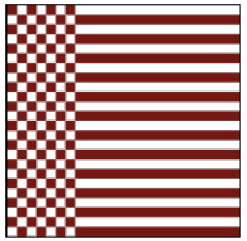
texture



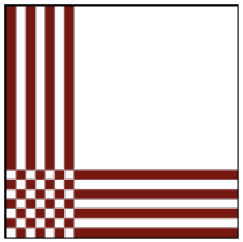
Wrap S : GL_REPEAT
Wrap T : GL_REPEAT



Wrap S : GL_REPEAT
Wrap T : GL_CLAMP



Wrap S : GL_CLAMP
Wrap T : GL_REPEAT



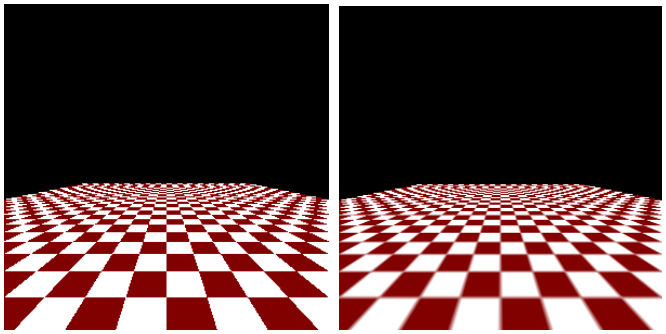
Wrap S : GL_CLAMP
Wrap T : GL_CLAMP

Step 2: Filter Modes

- Minification and magnification

Step 2: Mipmapping

- …

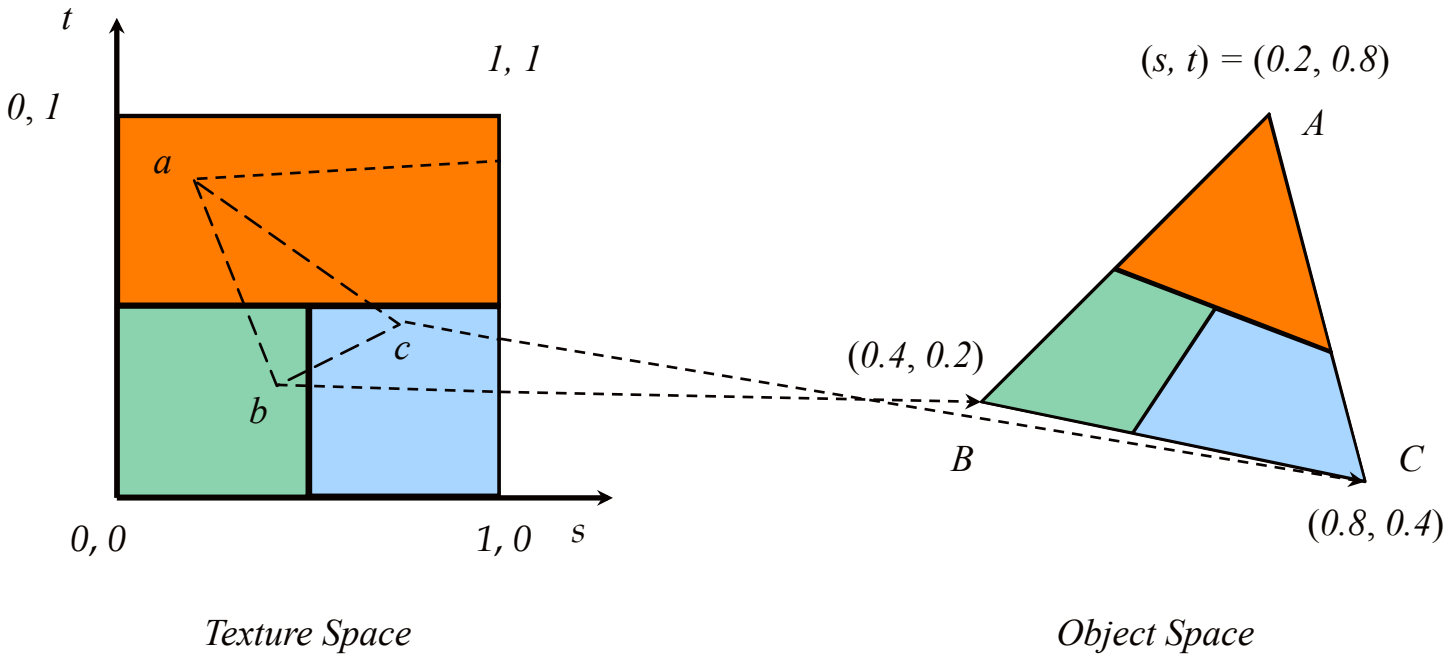


Step 3: Enable Texturing

- To enable/disable just call:
  - `glEnable(GL_TEXTURE_2D)`
  - `glDisable(GL_TEXTURE_2D);`
- What does texture mapping affect?
  - the current shading color of a pixel (after lighting) is multiplied by the corresponding texture color
- So, if the object is a near white color (0.8, 0.8, 0.8) at some point and the current texture color at that point is red (1, 0, 0), then when multiplied, it produces (0.8, 0, 0)

Step 4: Mapping a Texture

- Assign the texture coordinates
- `glTexCoord*()` is specified at each vertex

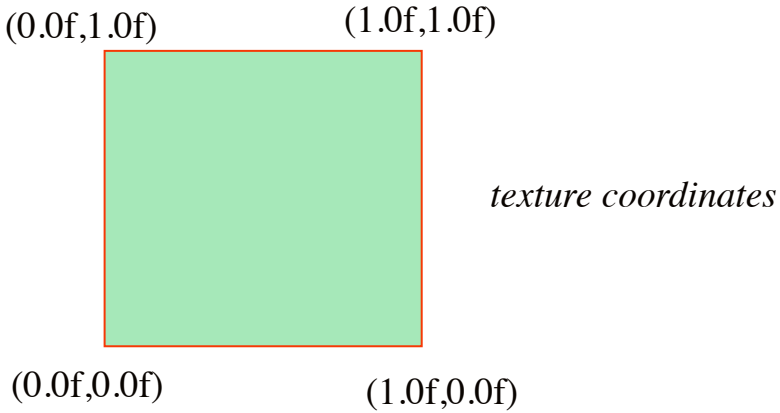


Step 4:
Mapping a Texture: *example*

- How to texture a quad? The following code assumes that texturing has been enabled and that there has been a texture uploaded with the id of 13.

```
glBindTexture (GL_TEXTURE_2D, 13);

glBegin (GL_QUADS);
    glTexCoord2f (0.0, 0.0); glVertex3f (0.0, 0.0, 0.0);
    glTexCoord2f (1.0, 0.0); glVertex3f (10.0, 0.0, 0.0);
    glTexCoord2f (1.0, 1.0); glVertex3f (10.0, 10.0, 0.0);
    glTexCoord2f (0.0, 1.0); glVertex3f (0.0, 10.0, 0.0);
glEnd ();
```

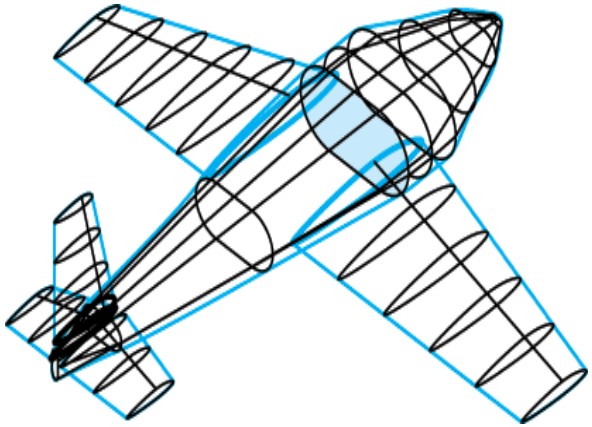


Texturing Mapping in OpenGL: summing up

- We have seen how a 2D texture image can be mapped to an object, at the rendering stage
  - for a polygon, we pin texture to vertices and interpolate (correctly!) at scan conversion time

- The texture value is used to modify the colour that would otherwise be drawn
  - options include replacing completely, or modulating (e.g. by multiplying shaded value with texture value)

What about complex 3D objects?

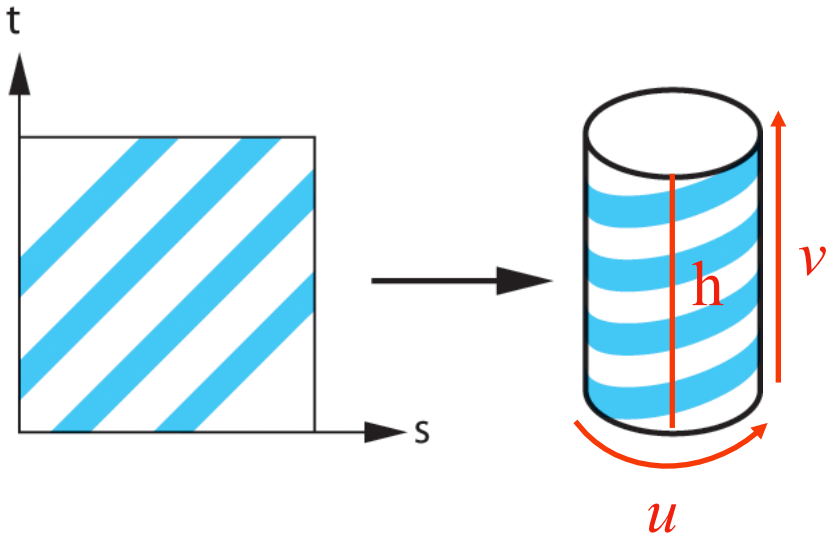


- It is easy to set texture coordinate for a single 2D polygon, but can be difficult to set the texture coordinates for complex 3D regions or objects.
- Besides:
  - in rendering based on pixel-to-pixel approach, the inverse mapping from screen coordinates to texture coordinates is needed
  - because of shading, mapping areas-to-areas and not point-to-point is required, which causes antialising problems, moire patterns etc.
- ***Two-Stage Mapping****:* An automatic solution to the mapping problem is to first map the texture to a simple intermediate surface **then** map the simple intermediate surface to the target surface

Automatic: cylindrical mapping

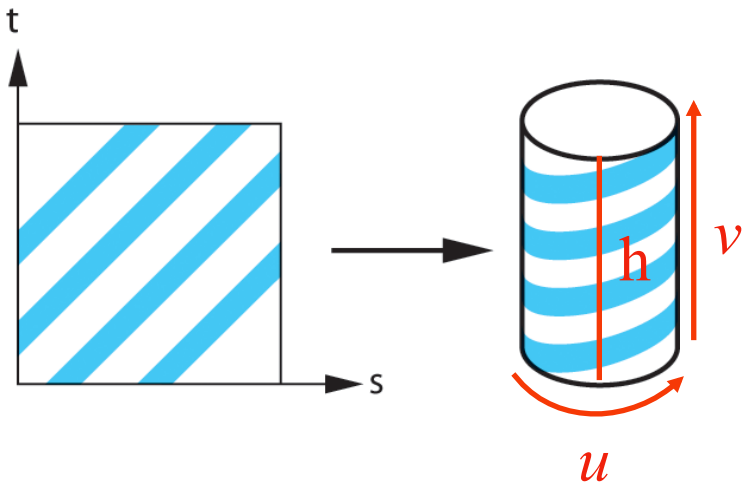
- Example: first map to a cylinder
- Like wrapping a label around a can of soup

- Convert rectangular coordinates (x, y, z) to cylindrical (r, , h), use only (h, μ) to index texture image



Automatic: cylindrical mapping

- Example: first map to a cylinder
- Like wrapping a label around a can of soup



- Convert rectangular coordinates (x, y, z) to cylindrical (r, θ, h), use only (h, θ) to index texture image

- Parametric cylinder:

$$x = r \cos(2\pi u)$$
$$y = v/\mathrm{h}$$
$$z = r \sin(2\pi u)$$

- Maps rectangle in u, v space to cylinder of radius r and height h in world coordinates:

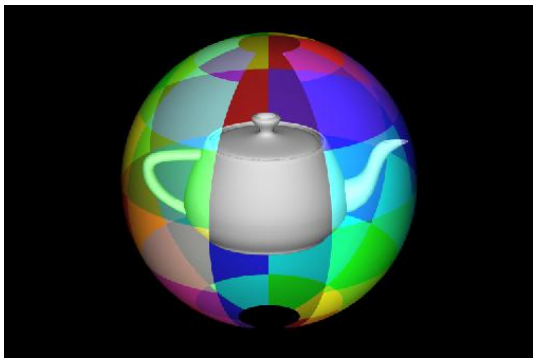
$$s = u$$
$$t = v$$

Automatic: spherical mapping

- Example: first map to a sphere
- Convert rectangular coordinates (x,y,z) to spherical (θ,ϕ)
- Parametric sphere:

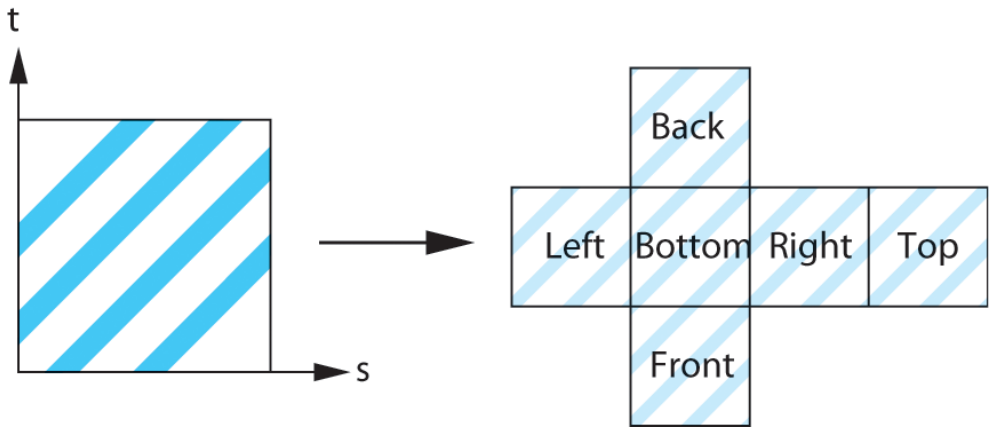
$$x = r \cos(2\pi u)$$
$$y = r \sin(2\pi u) \cos(2\pi v)$$
$$z = r \sin(2\pi u) \sin(2\pi v)$$

- For example: paste a world map onto a sphere to model the earth. But in the case of the sphere there is distortion at the poles (north and south)



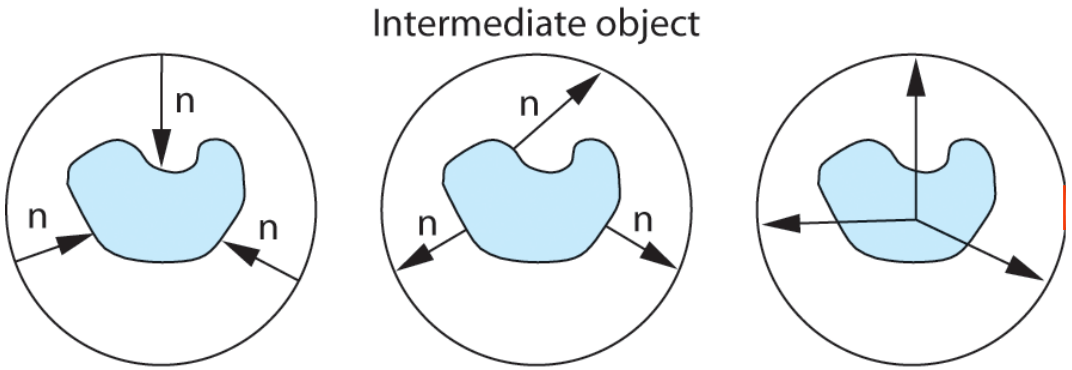
Automatic: box mapping

- Example: first map to a box



Automatic: stage-two mapping

- Now, we still need to map from an intermediate object (sphere, cylinder, or box) to the target object
  1. Intersect the normals from intermediate surface to target surface
  2. Intersect the normals from target surface to intermediate surface
  3. Intersect vectors from center of target surface to intermediate

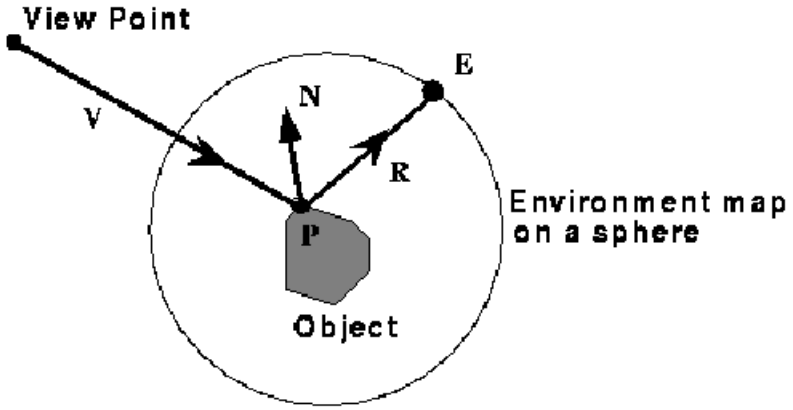


Mapping Techniques

- Texture Mapping
- **Environmental Mapping**
- Bump Mapping
- Light Mapping

Environmental Maps

- Use texture to represent reflected color
  - Texture indexed by reflection vector
  - Approximation works when objects are far away from the reflective object
- Environment mapping produces reflections on shiny objects
- Texture is transferred in the direction of the reflected ray from the environment map onto the object
- Reflected ray: $\boldsymbol{R}=2(\boldsymbol{N}\cdot\boldsymbol{V})\boldsymbol{N}-\boldsymbol{V}$
- What is in the map?



spherical map

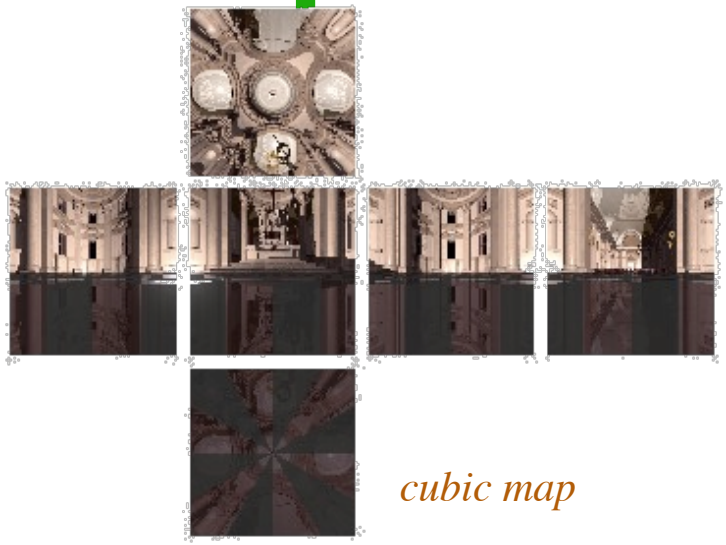
Approximations Made

- The map should contain a view of the world with the point of interest on the object as the eye
  - We can't store a separate map for each point, so one map is used with the eye at the center of the object
  - Introduces distortions in the reflection, but the eye doesn't notice
  - Distortions are minimized for a small object in a large room
- The object will not reflect itself
- The mapping can be computed at each pixel, or only at the vertices

cubic map

Types of Environment Maps

- The environment map may take one of several forms:
  - Cubic mapping
    - Easy to produce with rendering system
    - Possible to produce from phtographs
    - “uniform” resolution
    - Sim ple texture coordinates calculation
  - Spherical mapping (two variants)
    - Spatially variant resolution
  - Parabolic mapping
- Describes the shape of the surface on which the map “resides”
- Determines how the map is generated and how it is indexed
- What are some of the issues in choosing the map?



cubic map

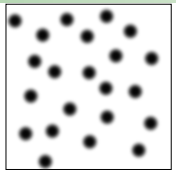
spherical maps

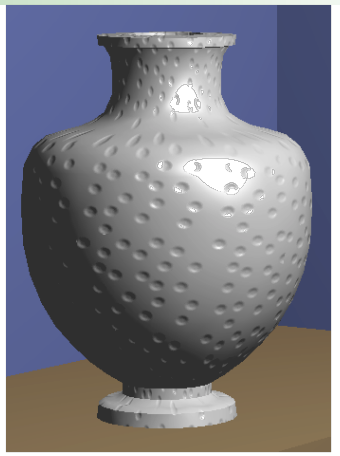
Mapping Techniques

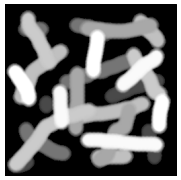
- Texture Mapping
- Environmental Mapping
- **Bump Mapping**
- Light Mapping

Bump Mapping

- This is another texturing technique
- Aims to simulate a dimpled or wrinkled surface
  - for example, surface of an orange
- Like Gouraud and Phong shading, it is a trick
  - surface stays the same
  - but the true normal is perturbed, or jittered, to give the illusion of surface ‘bumps’

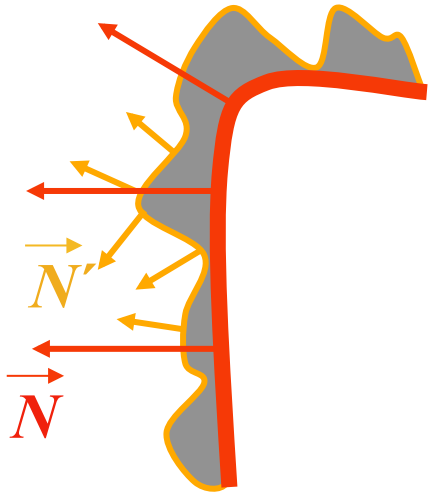






Bump Mapping: how does it work?

- To create a bump-like effect, we use texture to perturb normals
- Many textures are the result of small perturbations in the surface geometry
- Modeling these changes would result in an explosion in the number of geometric primitives.
- Bump mapping attempts to alter the lighting across a polygon to provide the illusion of texture.
- We can model this as deviations from some base surface.
- The question is then how these deviations change the lighting.



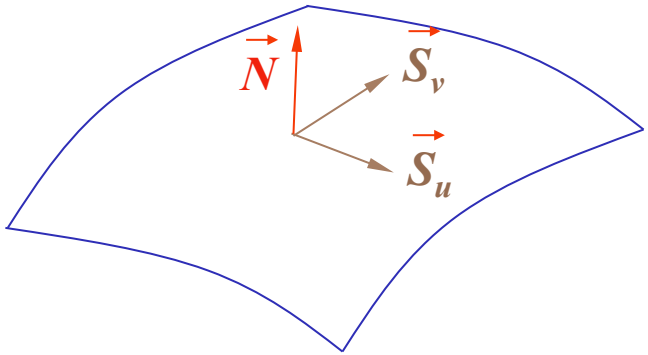
$$S(u,v) + B(u,v) = S'(u,v)$$

original surface	*bump map*	*bumped surface*

Bump Mapping: step 1 and 2

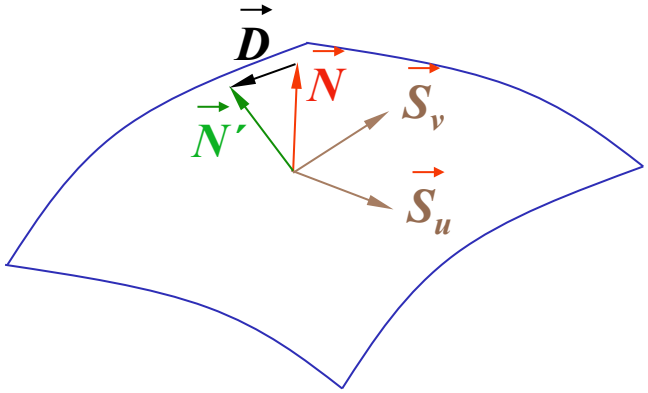
- Step 1: Putting everything into the same coordinate frame as B (u,v).
  - x(u,v), y(u,v), z(u,v) – this is given for parametric surfaces, but easy to derive for other analytical surfaces.
  - Or $\boldsymbol{S}(u,v)$
- Step 2: Define the tangent plane to the surface at a point (u,v) by using the two vectors $\vec{S}_u$ and $\vec{S}_v$.
  - The normal is then given by:

$$\vec{N} = \vec{S}_u \times \vec{S}_v$$



Bump Mapping: step 3, 4, and 5

- Step 3: The new surface positions are then given by:
  - $S'(u,v) = S(u,v) + B(u,v)\,\vec{N}$
  - where, $\vec{N} = \vec{N} / |\vec{N}|$



- Step 4: Differentiating leads to:
  - $\vec{S'}_u = \vec{S}_u + B_u \vec{N} + B(\vec{N})_u \approx \vec{S'}_u = \vec{S}_u + B_u \vec{N}$
  - $\vec{S'}_v = \vec{S}_v + B_v \vec{N} + B(\vec{N})_v \approx \vec{S'}_v = \vec{S}_v + B_v \vec{N}$

  If B is small.
- Step 5: This leads to a new normal:
  - $\vec{N'}(u,v) = \vec{S}_u \times \vec{S}_v - B_u(\vec{N} \times \vec{S}_v) + B_v(\vec{N} \times \vec{S}_u) + B_u B_v(\vec{N} \times \vec{N})$
    - $= \vec{N} - B_u(\vec{N} \times \vec{S}_v) + B_v(\vec{N} \times \vec{S}_u)$
    - $= \vec{N} + \vec{D}$

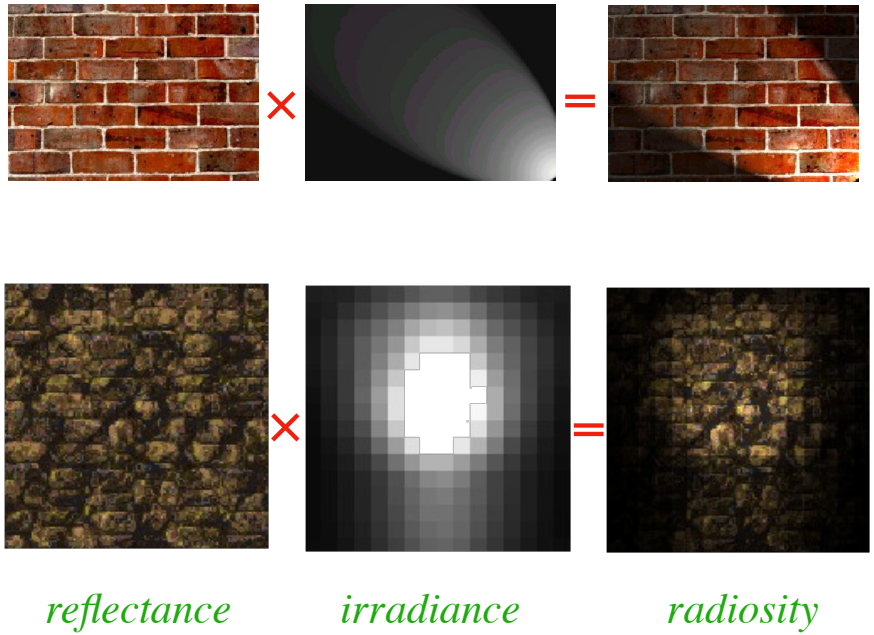
  - For efficiency, can store B_u and B_v in a 2-component texture map.
  - The cross products are geometry terms only.
  - $\vec{N'}$ will of course need to be normalized after the calculation and before lighting. This floating point square root and division makes it difficult to embed into hardware.

Mapping Techniques

- Texture Mapping
- Environmental Mapping
- Bump Mapping
- Light Mapping

Light Mapping

- Gouraud shading is established technique for rendering but has well known limitations
  - Vertex lighting only works well for small polygons…
  - … but we don't want lots of polygons!
- Solution is to pre-compute some canonical light effects as texture maps
- For example…
- Suppose we want to show effect of a wall light
  - Create wall as a single polygon
  - Apply vertex lighting
  - Apply texture map
  - In a second rendering pass, apply light map to the wall



Light Mapping

- Widely used in games industry
- Latest graphics cards will allow multiple texture maps per pixel