

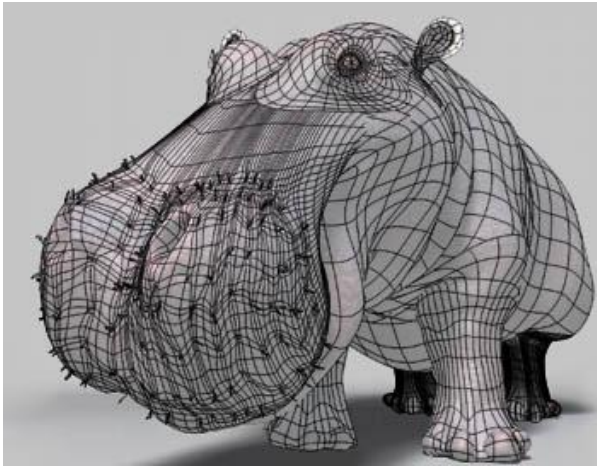
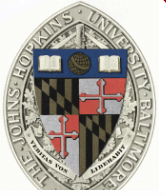


Texture Mapping

Misha Kazhdan

(601.457/657)

Textures



[J. Birn]

We know how to render geometry.

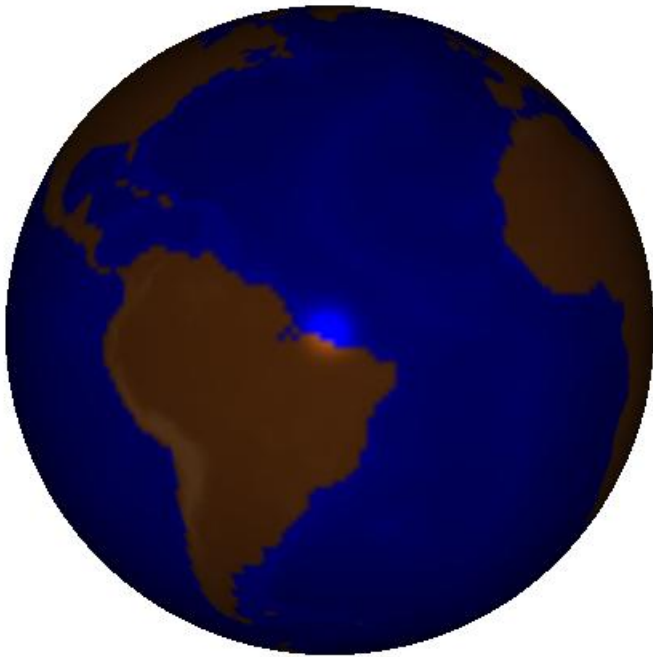
How do we get:

- Detailed geometry?
- Detailed colors?
- Shadows?



Textures

How should we draw surfaces with complex detail?



Target Model

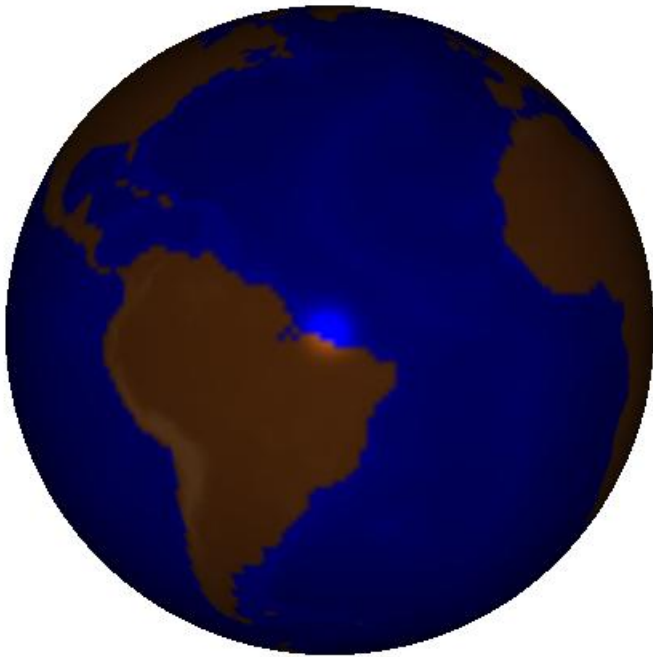


Textures

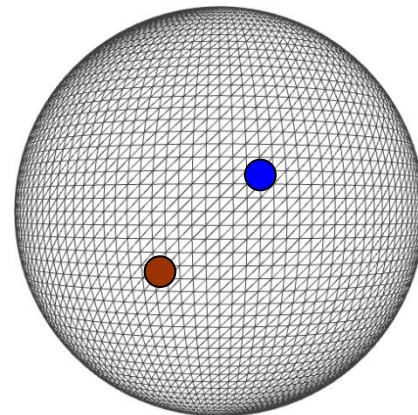
How should we draw surfaces with complex detail?

Direct:

- Tessellate finely and assign material properties to each vertex



Target Model



Complex Surface

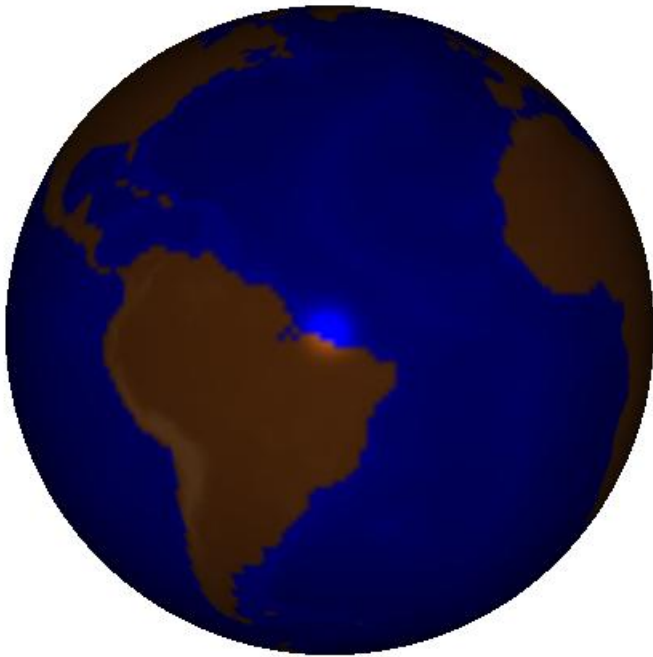


Textures

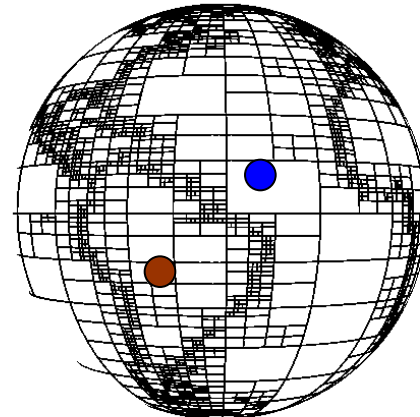
How should we draw surfaces with complex detail?

Direct:

- Tessellate adaptively and assign material properties to each vertex



Target Model



Complex Surface

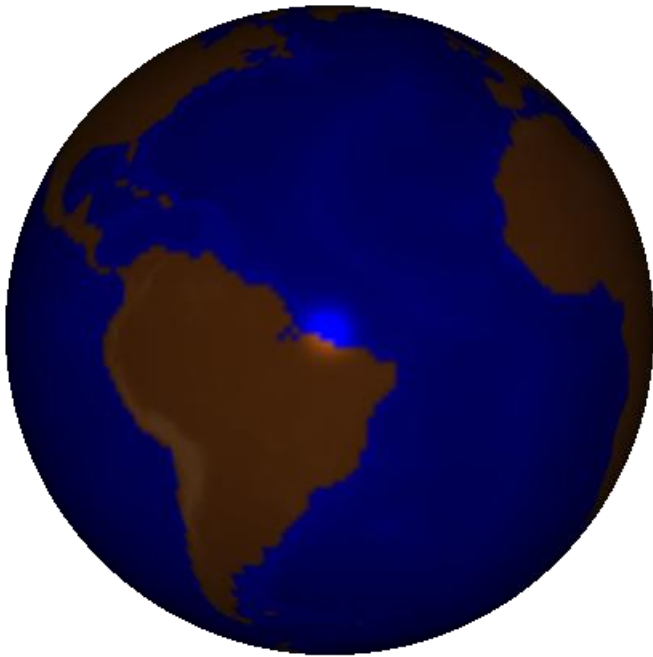


Textures

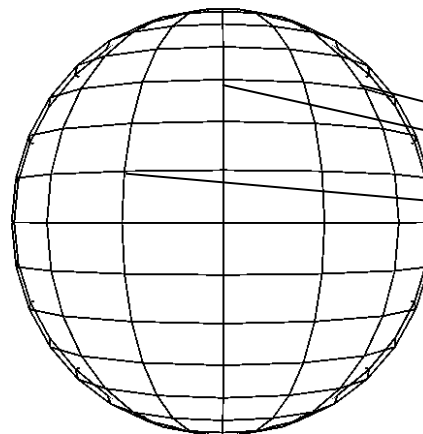
How should we draw surfaces with complex detail?

Indirect:

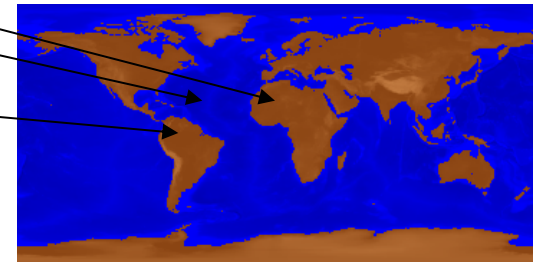
- Use a simple tessellation with an auxiliary *texture image*.
- Use *texture coordinates* stored at surface points to look up color values from the texture.



Target Model



Simple Surface



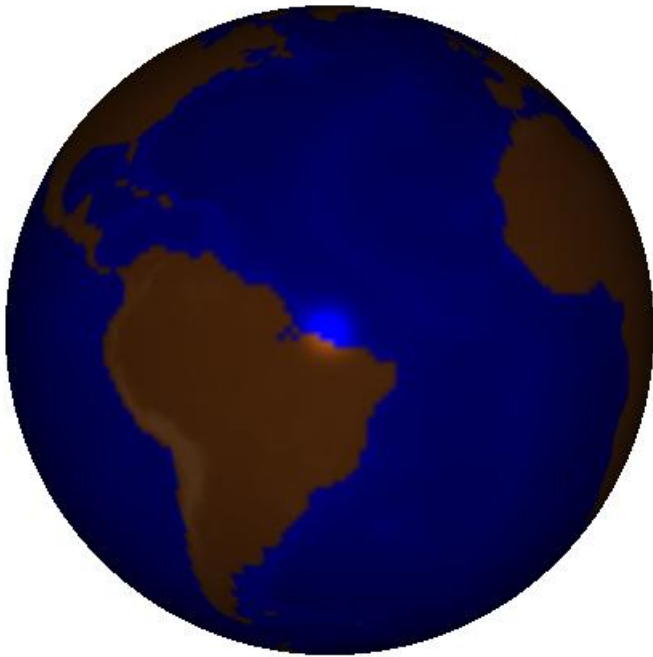
Texture Image



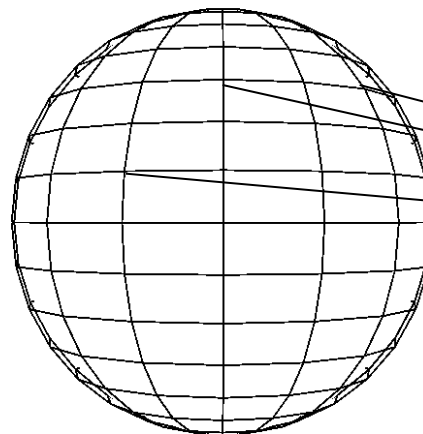
Textures

Advantages:

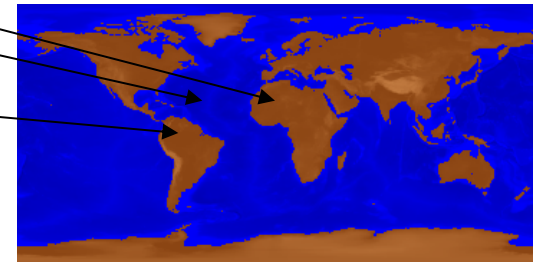
- The 3D geometry remains simple
- It is easier to design a texture than a surface signal



Target Model



Simple Surface



Texture Image



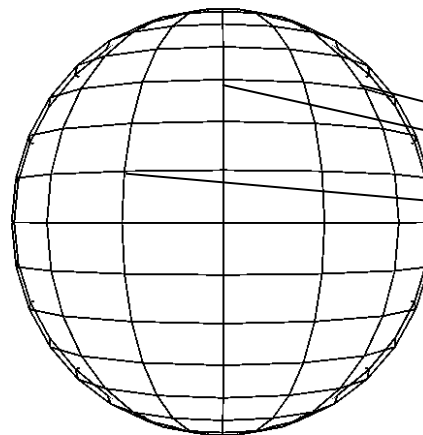
Textures (2 dimensions)

Implementation:

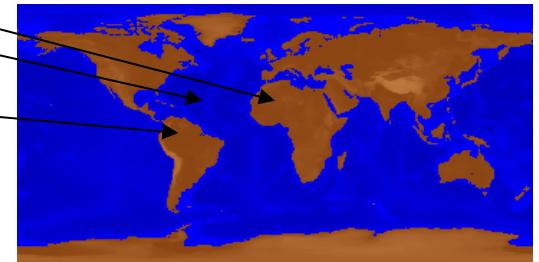
- Associate a *texture coordinate* to each vertex v :
 (s_v, t_v) with $(0 \leq s_v, t_v \leq 1)$
- When rasterizing, *interpolate* to get the texture coordinate at pixel p :
 (s_p, t_p)
- *Sample* the texture at (s_p, t_p) to get the color at p .

Terminology:

Texture elements
are called *texels*



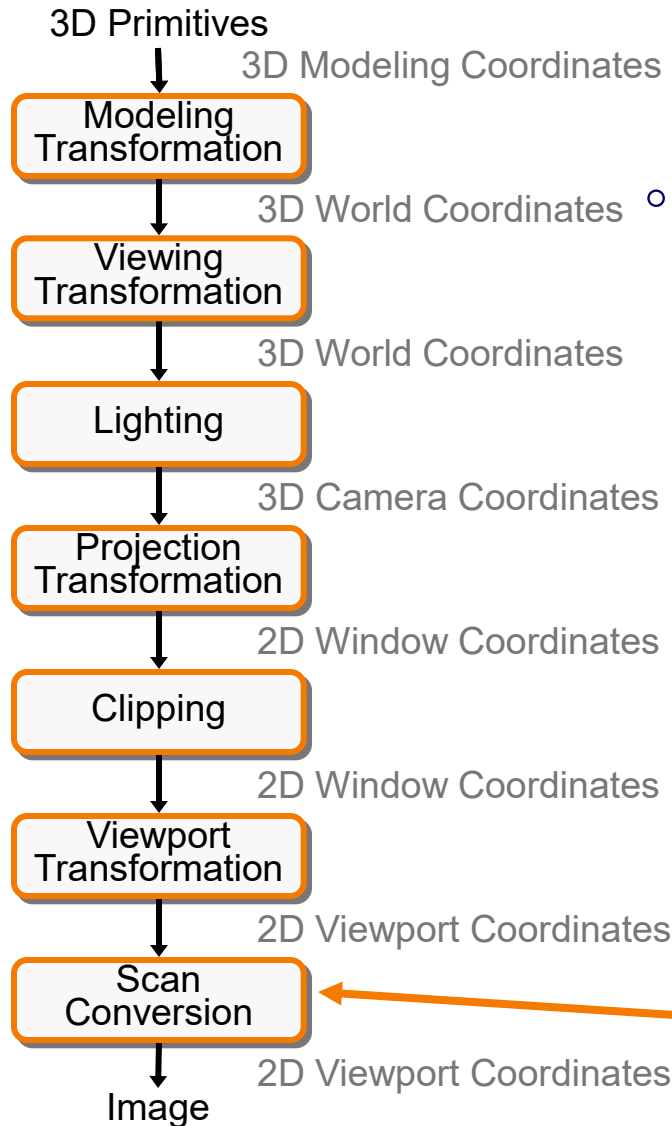
Simple Surface



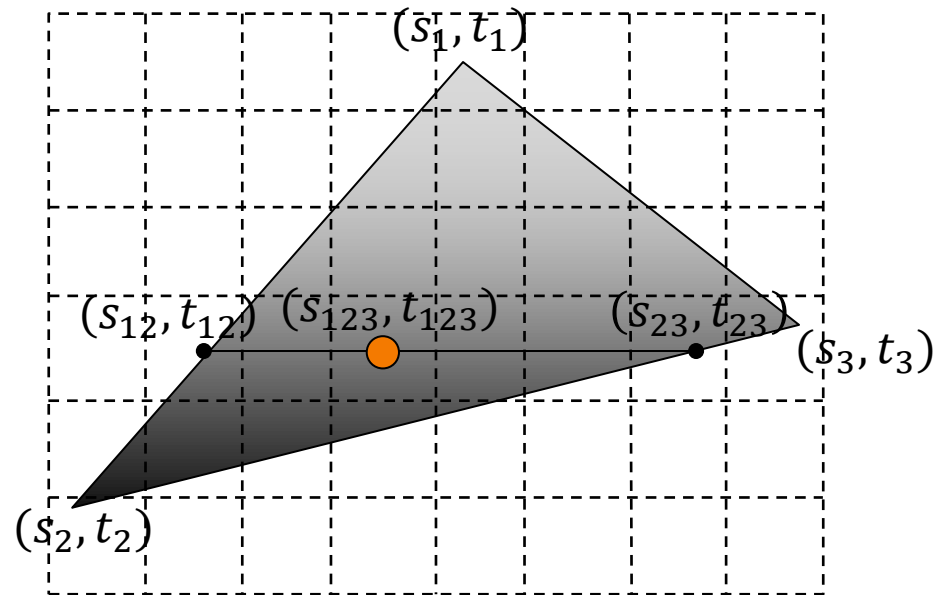
Texture Image



3D Rendering Pipeline



- Perform the **texture interpolation** and **look-up** while rasterizing the pixels.



texture mapping



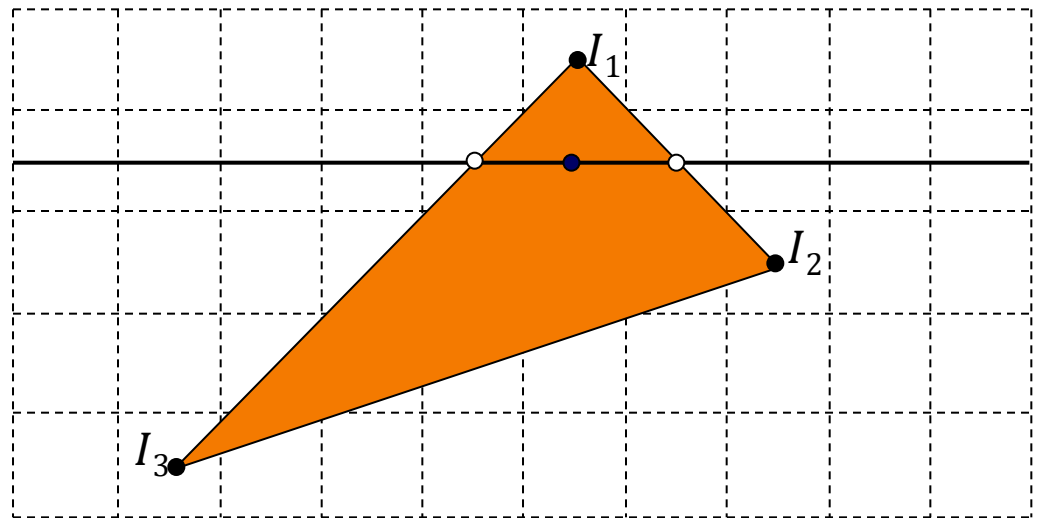
Scan Conversion

How should we average information (e.g. color, normal, depth, texture) from the triangle vertices?

- Interpolate using screen space (2D) weights
- Interpolate using world space (3D) weights

It's easier to do the interpolation in 2D.

Is there a difference?





Scan Conversion

Projective transformations (recall)

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Properties of projective transformations:

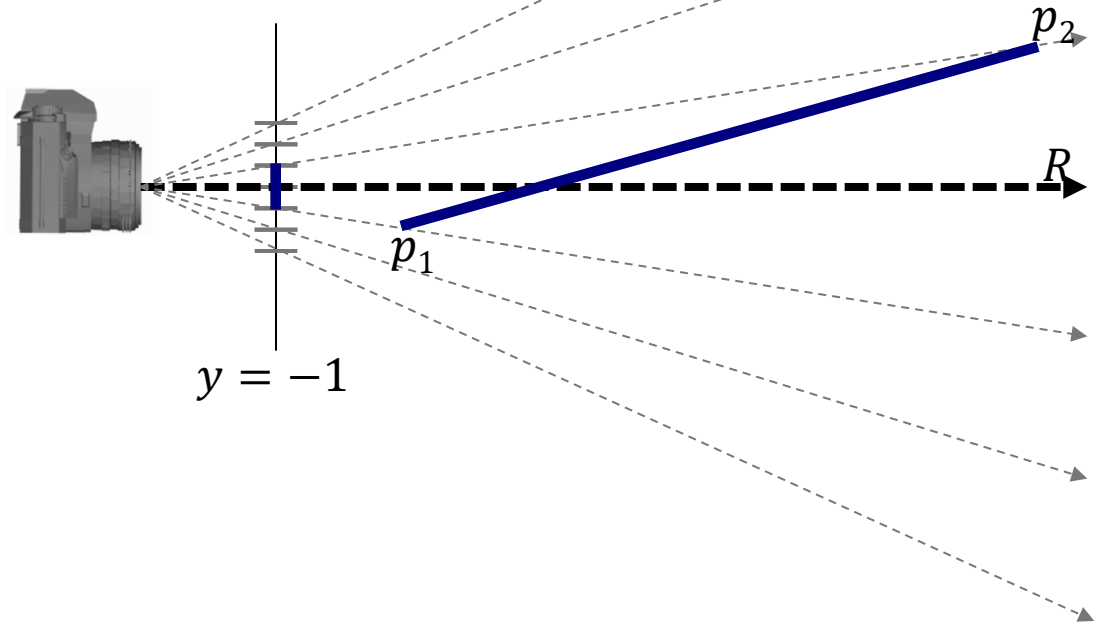
- Origin does not necessarily map to origin
- Lines map to lines
- (Weighted) average is not necessarily preserved
- Parallel lines do not necessarily remain parallel
- Closed under composition



Scan Conversion Example

A line segment in 2D projected onto a 1D window.

How to interpolate the information from vertices p_1 and p_2 at the pixel corresponding to ray R ?

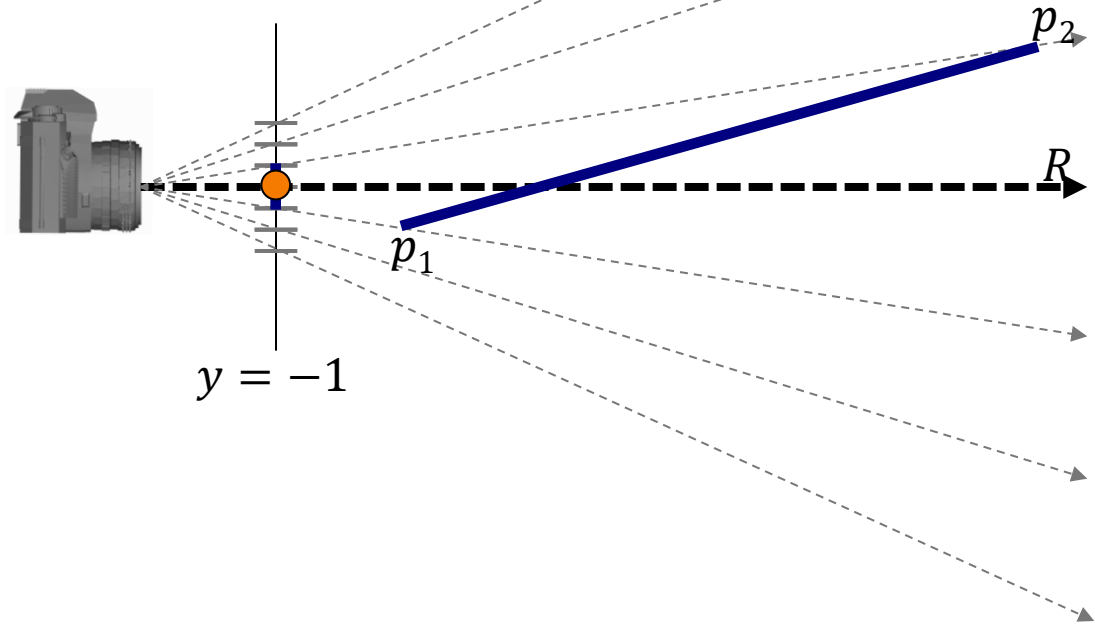




Scan Conversion Example

A line segment in 2D projected onto a 1D window.

1. [1D] The projected intersection of the ray with the line segment is directly between the projections of p_1 and p_2 :
⇒ Use equal contributions from p_1 and p_2 .

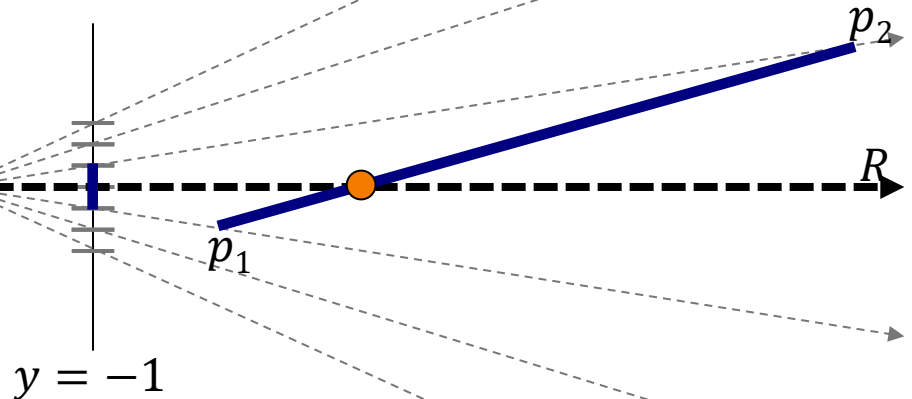
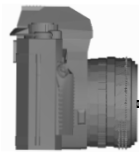




Scan Conversion Example

A line segment in 2D projected onto a 1D window.

1. [1D] The projected intersection of the ray with the line segment is directly between the projections of p_1 and p_2 :
⇒ Use equal contributions from p_1 and p_2 .
2. [2D] The intersection of the ray with the line segment is closer to p_1 than to p_2 :
⇒ Use more information from p_1 than from p_2 .





Scan Conversion Example

Q: How do we interpolate correctly at $(x, -1)$?

If $p_1 = (x_1, z_1)$ and $p_2 = (x_2, z_2)$, to find the blending value α for a pixel falling at position x in the screen we need to solve:

$$(1 - \alpha)(x_1, z_1) + \alpha(x_2, z_2) \sim (x, -1)$$



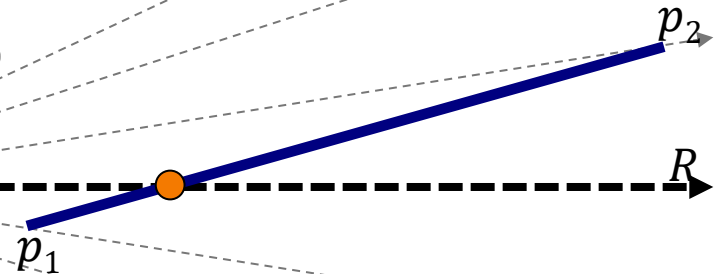
$$((1 - \alpha)x_1 + \alpha x_2, (1 - \alpha)z_1 + \alpha z_2) \sim (x, -1)$$



$$\frac{(1 - \alpha)x_1 + \alpha x_2}{(1 - \alpha)z_1 + \alpha z_2} = \frac{x}{-1}$$



$z = -1$





Scan Conversion Example

Q: How do we interpolate correctly at $(x, -1)$?

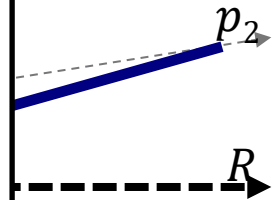
If $p_1 = (x_1, z_1)$ and $p_2 = (x_2, z_2)$, to find the blending value α for a pixel falling at position x in the screen we need to solve:

$(1 - \alpha)$ To compute the interpolation weights, perform a perspective divide:

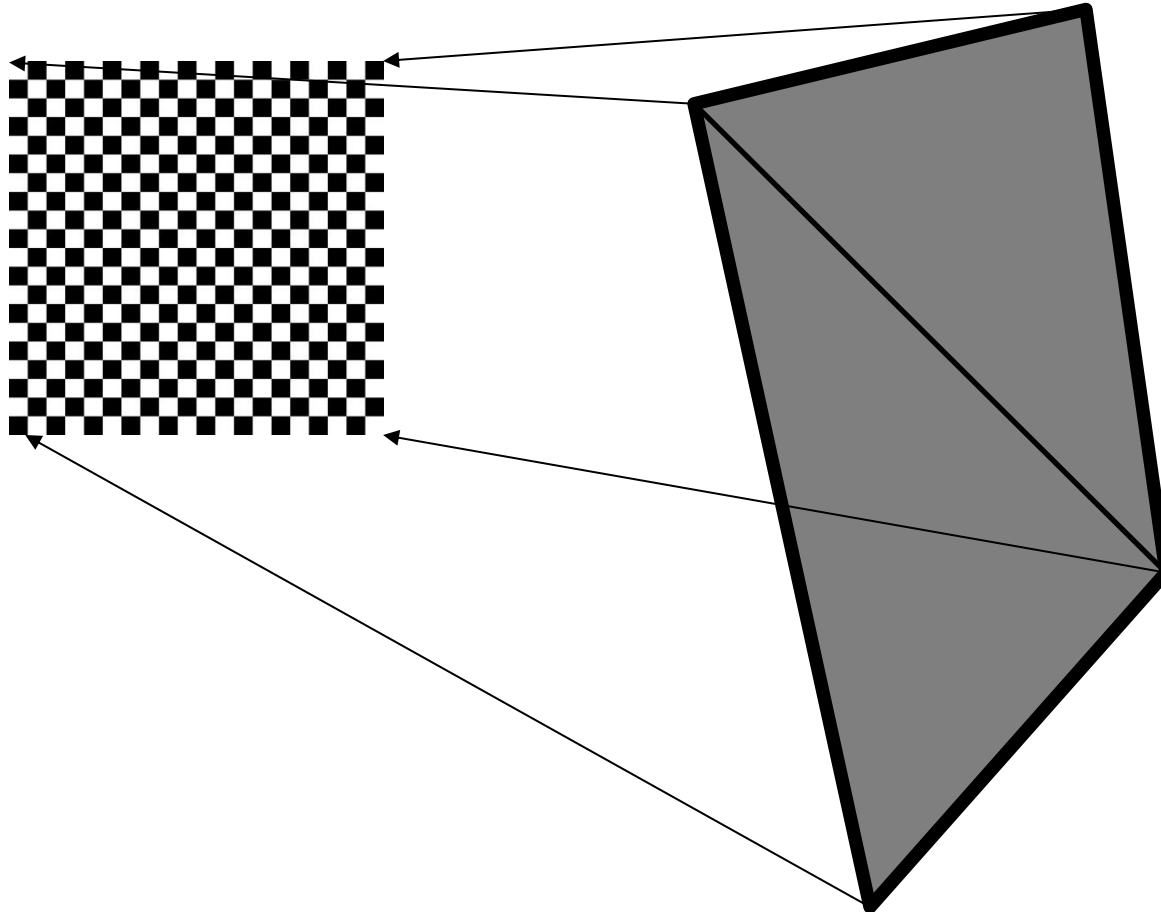
$$\frac{(1 - \alpha)x_1 + \alpha x_2}{(1 - \alpha)z_1 + \alpha z_2} = \frac{x}{-1}$$

This is different than solving for the blending value in the image plane:

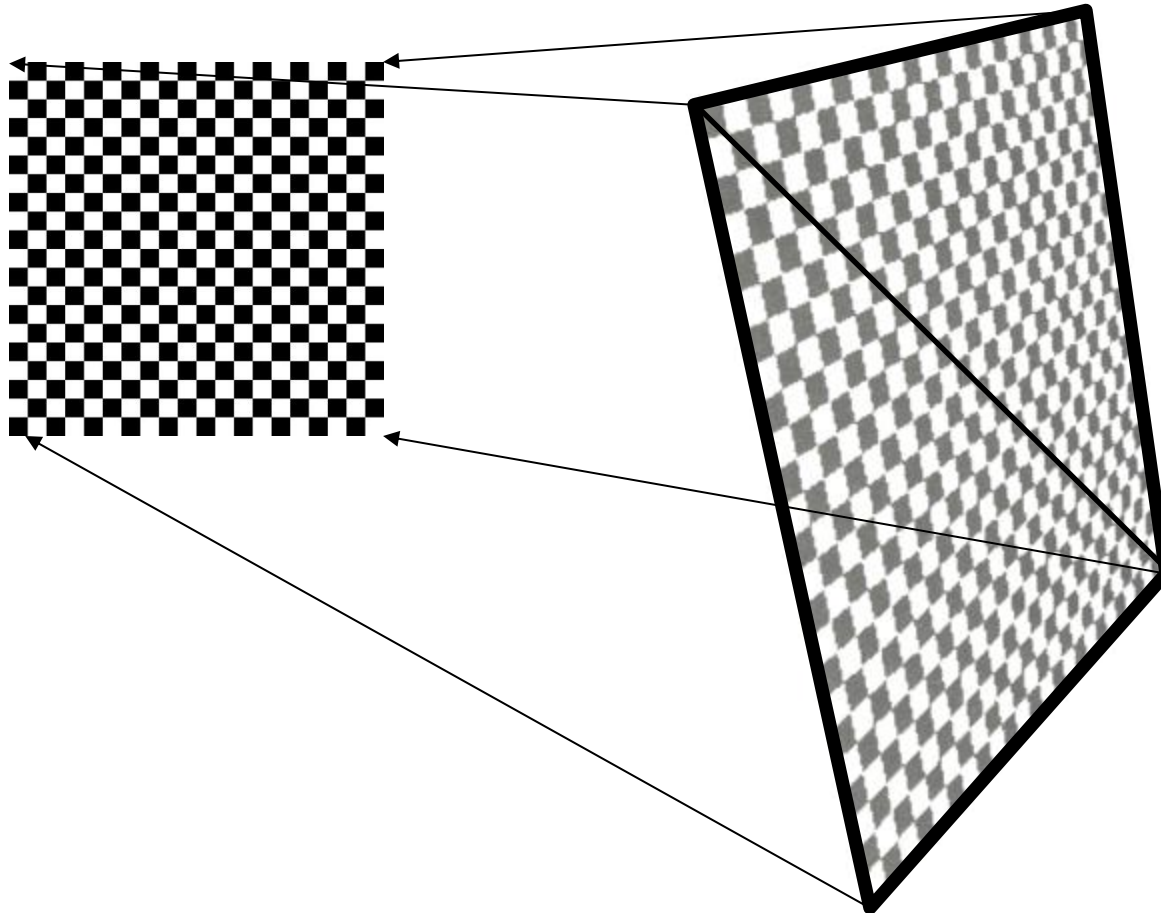
$$(1 - \alpha) \frac{x_1}{z_1} + \alpha \frac{x_2}{z_2} = \frac{x}{-1}$$



Texture Mapping



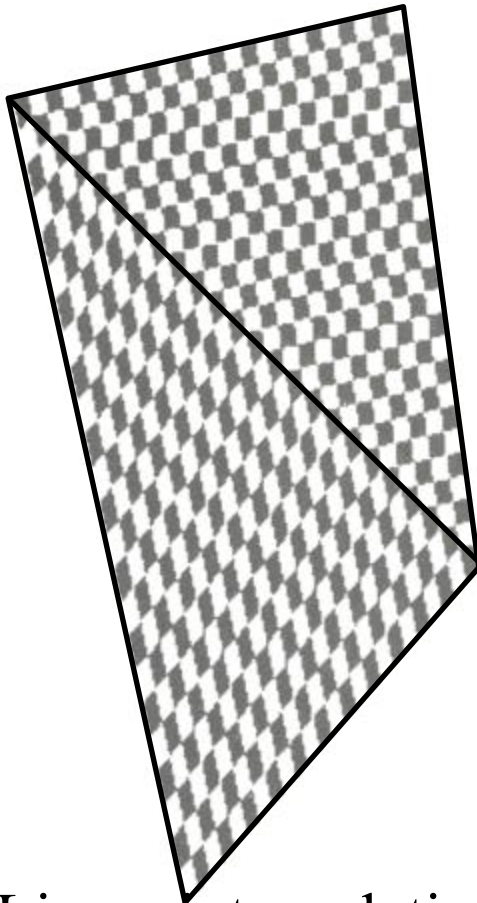
Texture Mapping



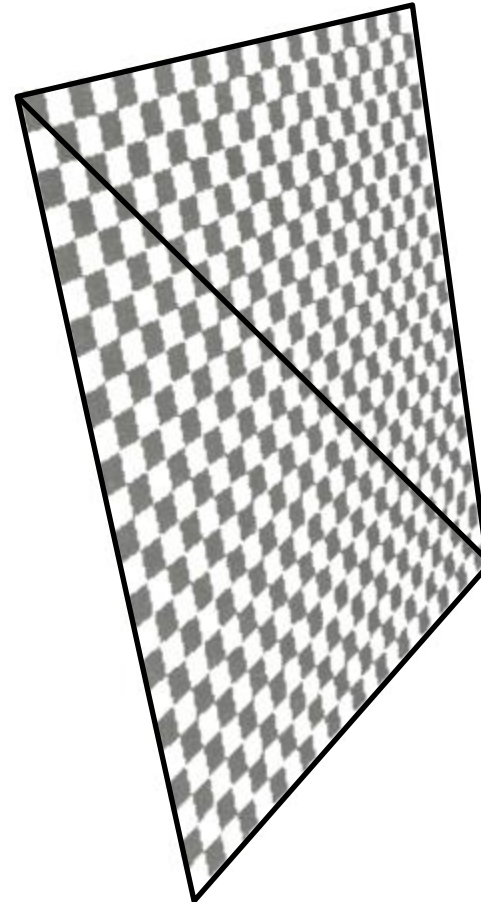
Correct interpolation
of texture coordinates
with perspective divide

Hill Figure 8.42

Texture Mapping



Linear interpolation
of texture coordinates
w/o perspective divide

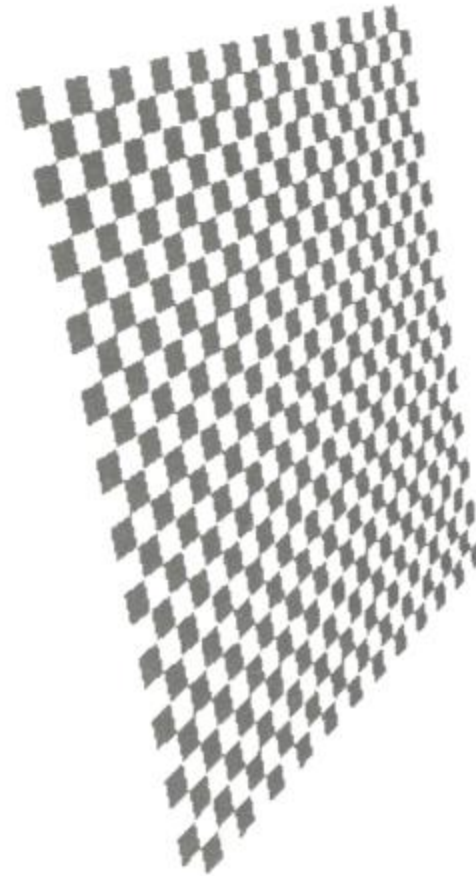


Correct interpolation
of texture coordinates
w/ perspective divide

Texture Mapping



Linear interpolation
of texture coordinates
w/o perspective divide



Correct interpolation
of texture coordinates
w/ perspective divide



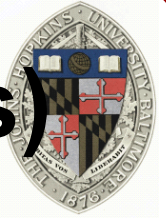
Overview

Texture mapping methods

- Parameterization
- Sampling

Texture mapping applications

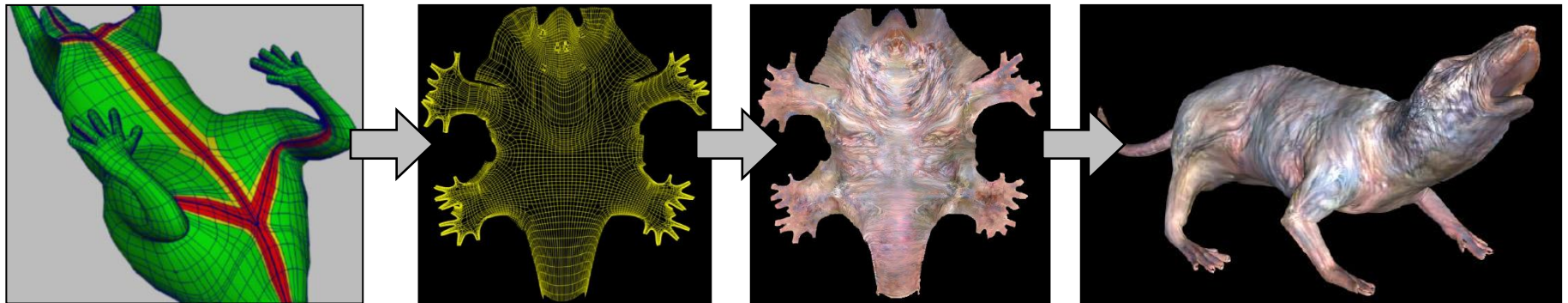
- Modulation textures
- Illumination mapping
- Bump mapping
- Environment mapping
- Shadow maps



Map to a 2D Domain (w/ Added Cuts)

- Introduce cuts to give the surface a disk topology
- Map the cut surface to the 2D plane
- Assign texture coordinates in the plane

-
- ✓ Good cut placement can reduce distortion
 - ✗ Need to ensure cross-seam continuity
 - ✗ Have to contend with distortion



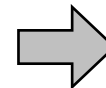
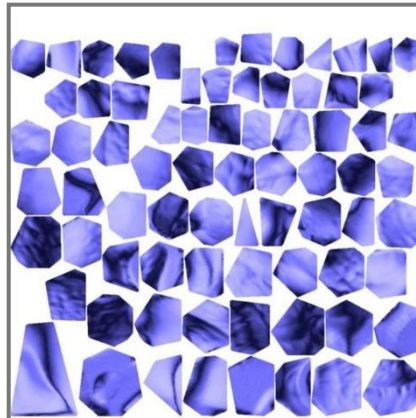
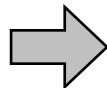
[Piponi, 2000]



Texture Atlases

- Decompose the surface into multiple charts
- Map each chart to the 2D plane
- Assign texture coordinates in the plane

-
- ✓ Less distortion in the mapping
 - ✗ Harder to ensure cross-seam continuity
 - ✗ Need to pack the atlases into 2D



[Sander, 2001]



Overview

Texture mapping methods

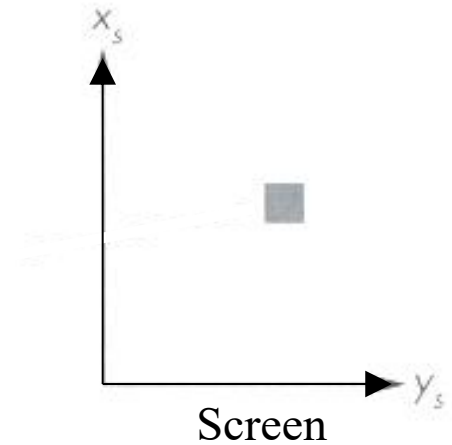
- Parameterization
- **Sampling**

Texture mapping applications

- Modulation textures
- Illumination mapping
- Bump mapping
- Environment mapping
- Shadow maps

Texture Filtering

Given pixel on a screen:

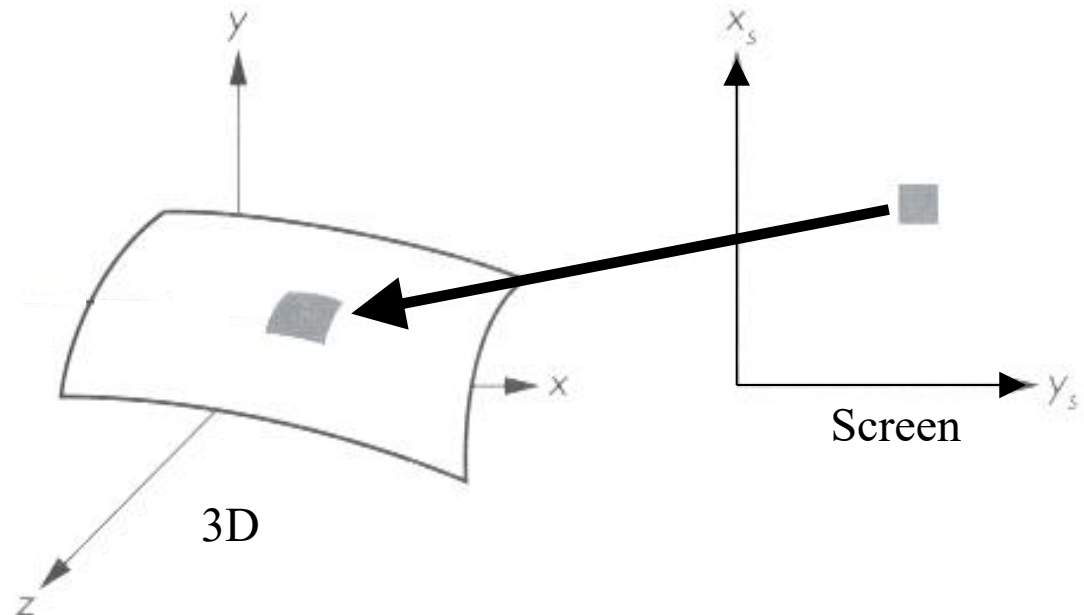




Texture Filtering

Given pixel on a screen:

1. Determine the corresponding surface patch



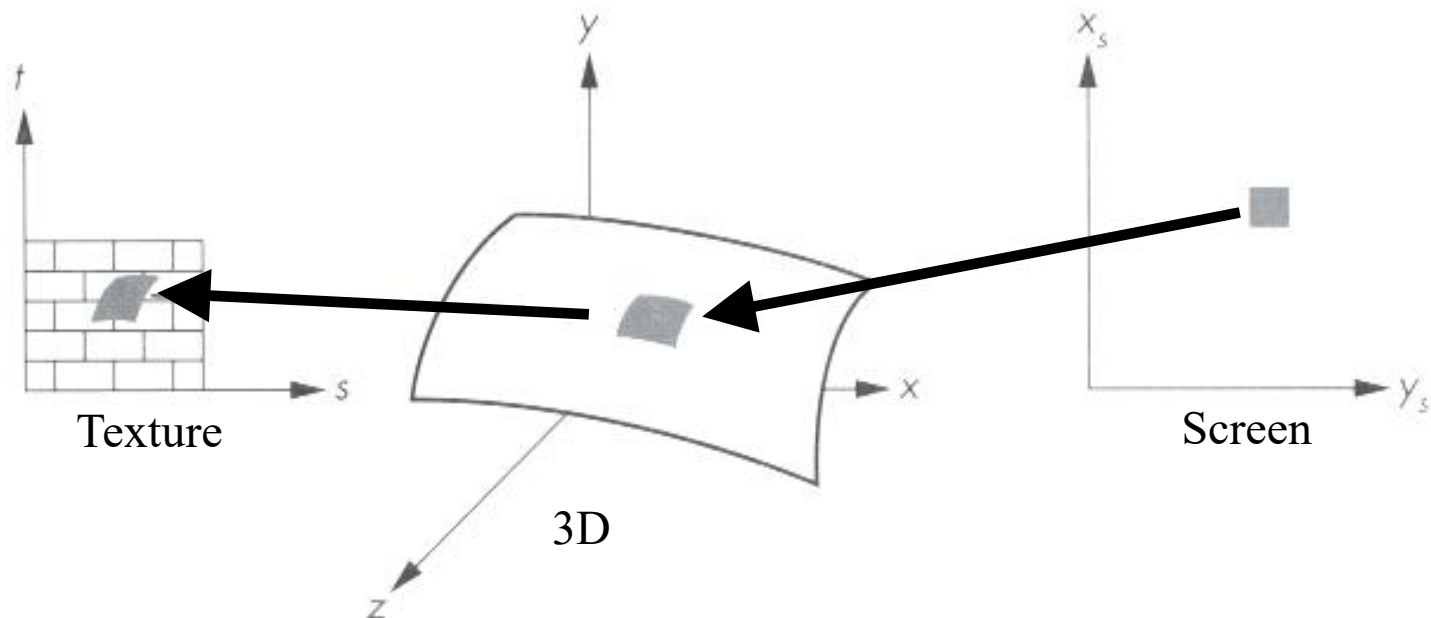
Angel Figure 9.4



Texture Filtering

Given pixel on a screen:

1. Determine the corresponding surface patch
2. Determine the corresponding texture patch

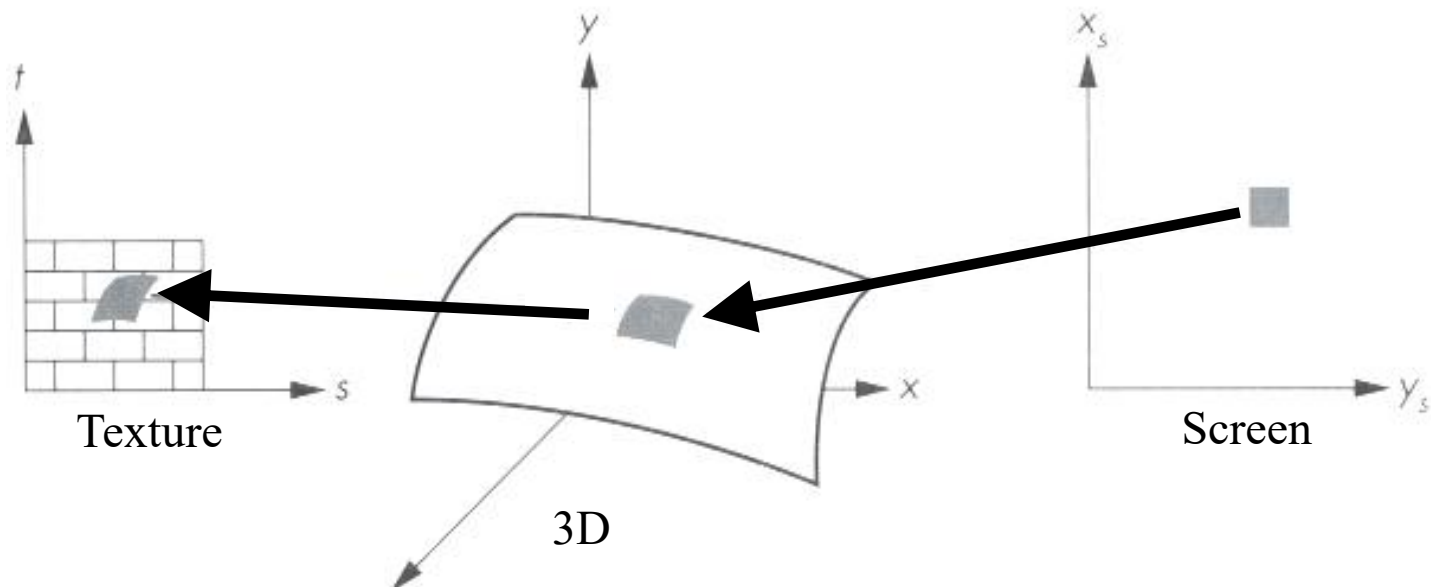




Texture Filtering

Given pixel on a screen:

1. Determine the corresponding surface patch
2. Determine the corresponding texture patch



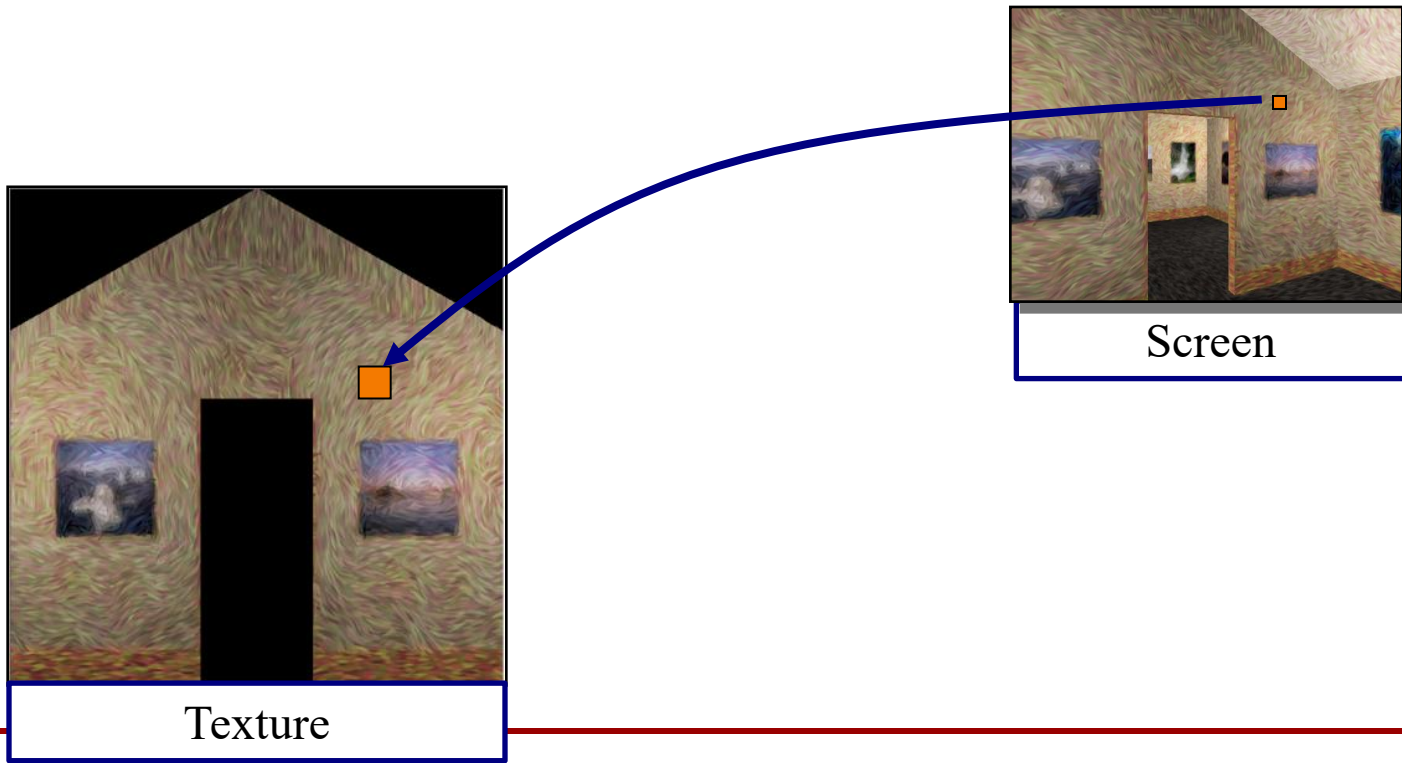
While the true shape of the texture patch can be hard to compute, we can approximate it by a parallelogram using the Jacobian.



Texture Filtering

Given pixel on a screen:

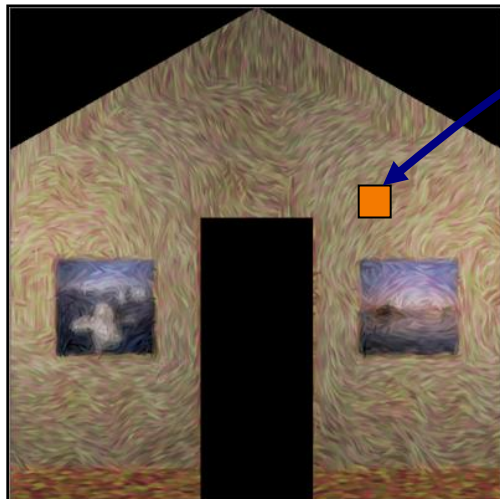
1. Determine the corresponding surface patch
2. Determine the corresponding texture patch
3. Average texel values over the texture patch





Texture Filtering

- ✖ Size of texture patch depends on the deformation
 - Computation is proportional to the pixel footprint
- Can pre-filter images for better performance
 - MIP (Multum In Parvo*) maps
 - Summed area tables



Texture

Average over
many pixels



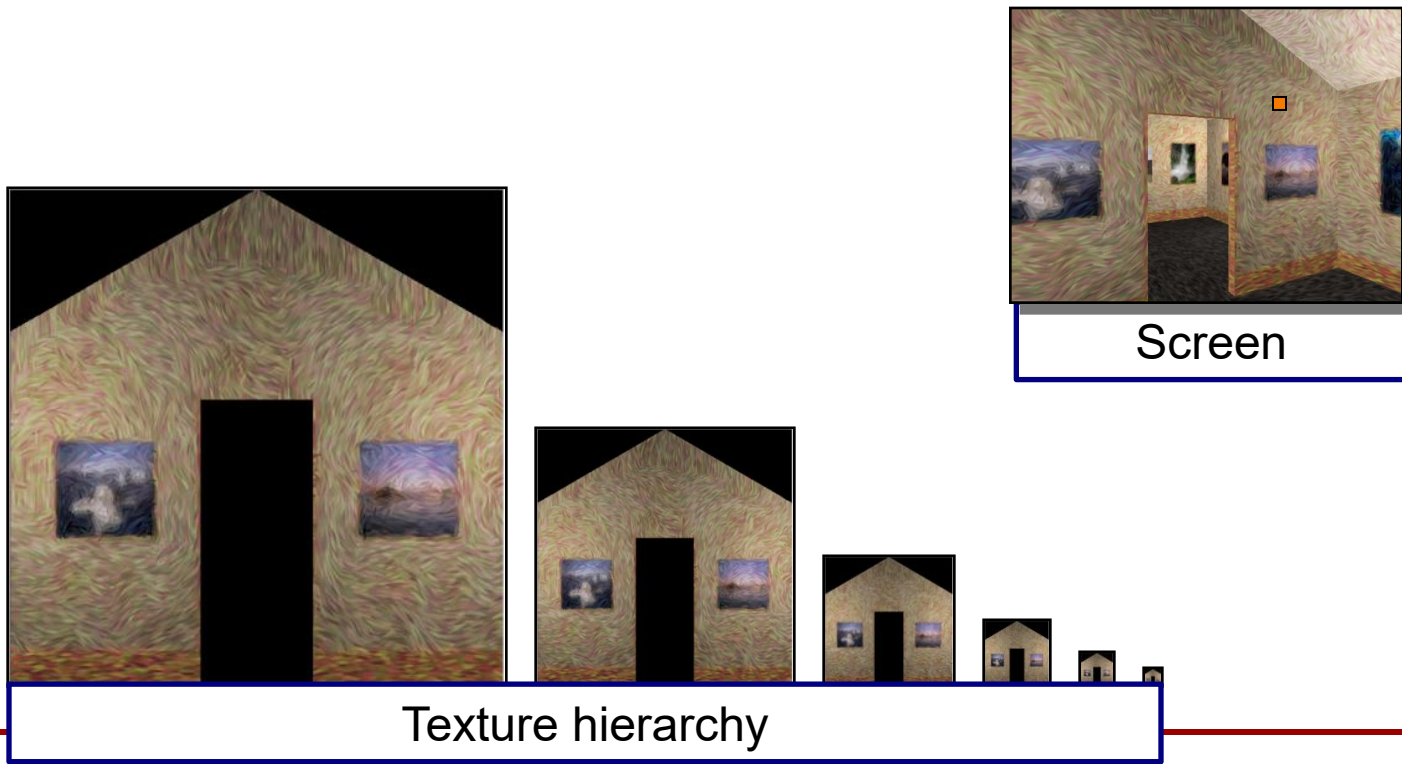
Screen

*multum in parvo = much in little



MIP Maps

Pre Processing: Compute a hierarchy of successively down-sampled texture images



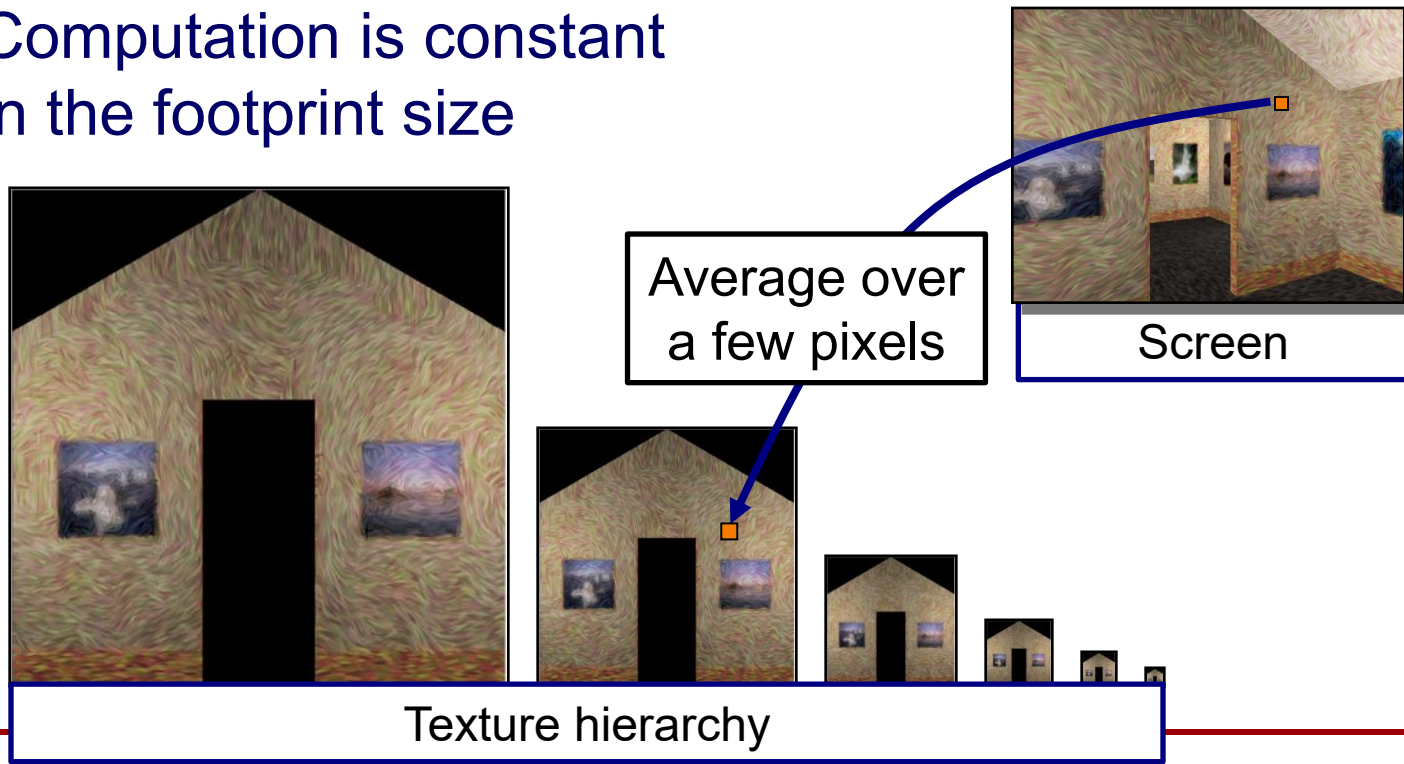


MIP Maps

Pre Processing: Compute a hierarchy of successively down-sampled texture images

Run-time: Sample the closest MIP map level(s)

- Easy for hardware
- Computation is constant in the footprint size

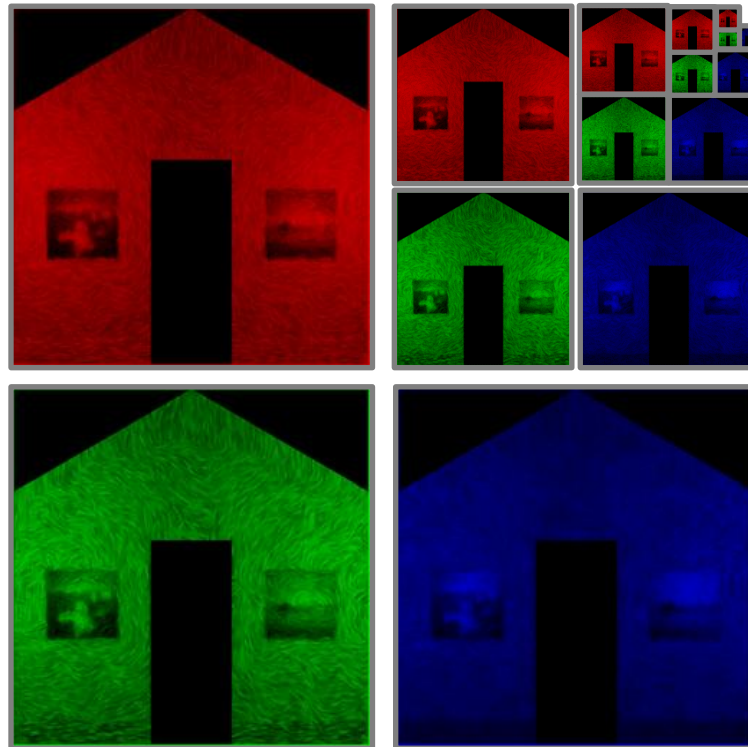




MIP Maps

Pre Processing: Compute a hierarchy of successively down-sampled texture images

✓ Storage is $\frac{4}{3} \times$ the size of the input image





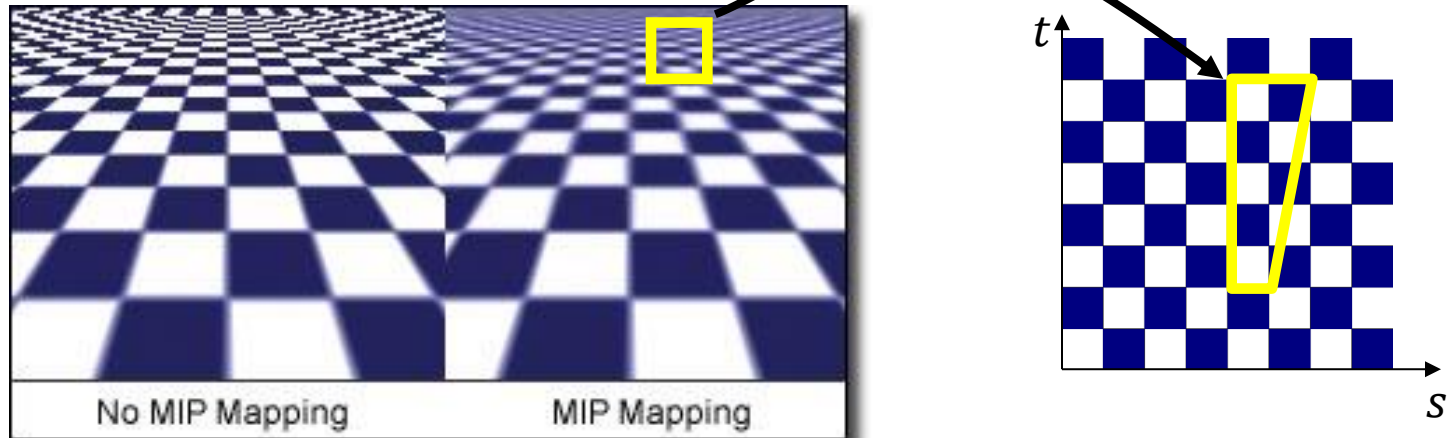
MIP Maps

Pre Processing: Compute a hierarchy of successively down-sampled texture images

✓ Storage is $4/3\times$ the size of the input image

Run-time: Sample the closest MIP map level(s)

✗ The filtering is isotropic – assumes identical compression along vertical and horizontal directions



[Trading aliasing for blurring!]

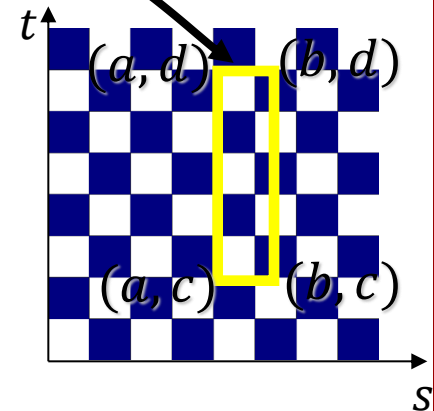
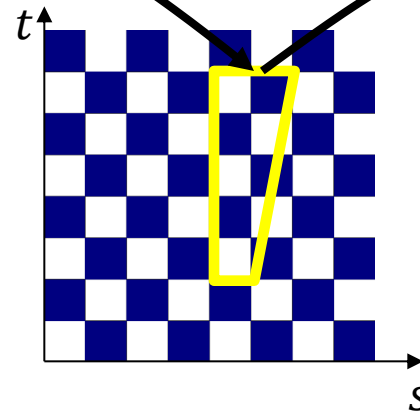
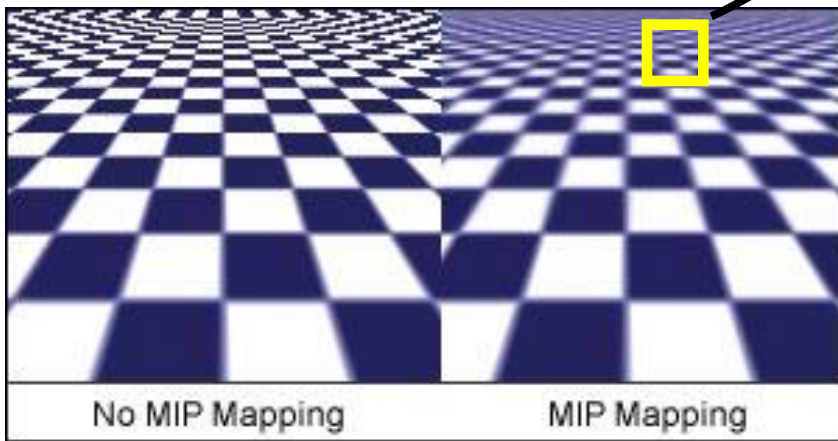


Summed-Area Tables

Key Idea:

Approximate the summation/integration over an arbitrary region by a summation/integration over an axis-aligned rectangle:

$$\text{Sum}([a, b] \times [c, d]) = \int_a^b \int_c^d f(x, y) dy dx$$



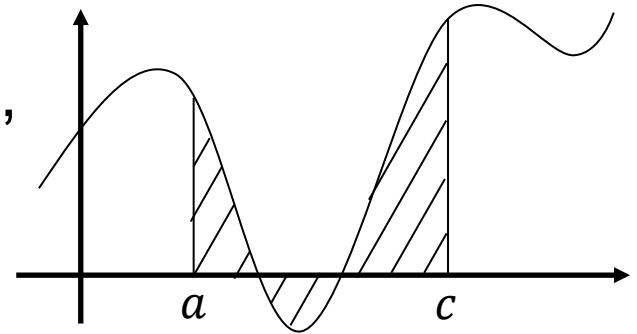


Summed-Area Tables (1D)

Integration:

Given a function $f(x)$ and interval $[a, c]$, the integral of f over the interval is:

$$\int_a^c f(x) dx$$



Naïve Approach:

Pre-compute $S(a, b) \equiv \int_a^b f(x) dx$ in a look-up table and evaluate that.

- ✓ Fast (constant time) look up
- ✗ Replaces 1D function f with 2D function S .

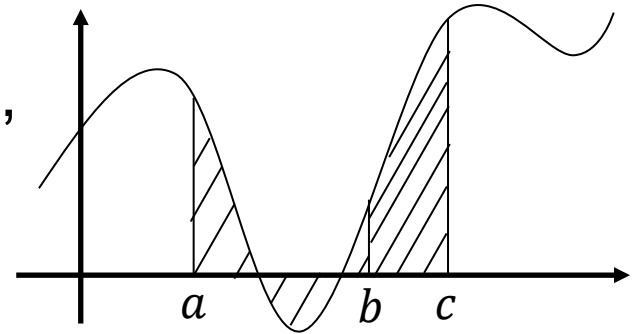


Summed-Area Tables (1D)

Integration:

Given a function $f(x)$ and interval $[a, c]$, the integral of f over the interval is:

$$\int_a^c f(x) dx$$



Recall:

For any point $b \in [a, c]$ in the interval, we have:

$$\int_a^c f(x) dx = \int_a^b f(x) dx + \int_b^c f(x) dx$$

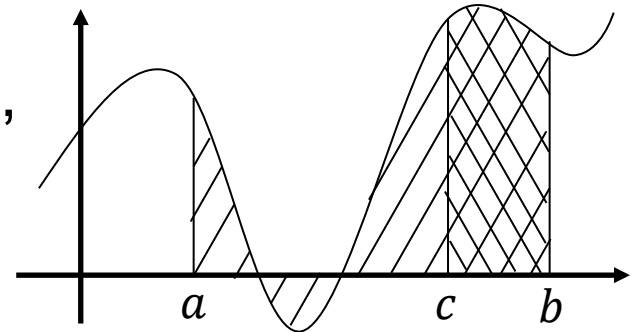


Summed-Area Tables (1D)

Integration:

Given a function $f(x)$ and interval $[a, c]$, the integral of f over the interval is:

$$\int_a^c f(x) dx$$



Recall:

For any point $b \in [a, c]$ in the interval, we have:

$$\int_a^c f(x) dx = \int_a^b f(x) dx + \int_b^c f(x) dx$$

This is true even if b is outside the interval since:

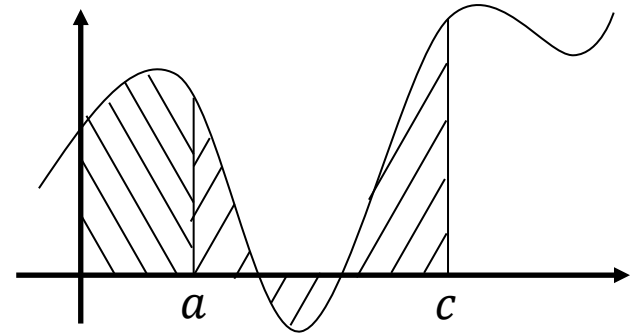
$$\int_a^b f(x) dx = - \int_b^a f(x) dx$$



Summed-Area Tables (1D)

Approach:

Replace the integral over an interval, with two variable end-points with the difference between integrals with one variable end-point:



$$\int_a^c f(x) dx = \int_{\textcircled{0}}^c f(x) dx - \int_{\textcircled{0}}^a f(x) dx$$

⇒ Replace a look-up in the 2D function $S(a, c) = \int_a^c f(x) dx$ with two look-ups in the 1D function $S_0(b) = \int_0^b f(x) dx$



Summed-Area Tables (2D)

Integration:

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\int_a^b \int_c^d f(s, t) dt ds$$



Summed-Area Tables (2D)

Integration:

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\int_a^b \left[\int_c^d f(s, t) dt \right] ds = \int_a^b \left(\left[\int_0^d f(s, t) dt \right] - \left[\int_0^c f(s, t) dt \right] \right) ds$$



Summed-Area Tables (2D)

Integration:

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\begin{aligned} \int_a^b \int_c^d f(s, t) dt ds &= \boxed{\int_a^b} \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds \\ &= \boxed{\int_0^b} \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds - \boxed{\int_0^a} \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds \end{aligned}$$



Summed-Area Tables (2D)

Integration:

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\begin{aligned}\int_a^b \int_c^d f(s, t) dt ds &= \int_a^b \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds \\ &= \int_0^b \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds - \int_0^a \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds \\ &= \underbrace{\int_0^b \int_0^d f(s, t) dt ds}_{\text{Summed-Area Table}} - \underbrace{\int_0^b \int_0^c f(s, t) dt ds}_{\text{Summed-Area Table}} - \underbrace{\int_0^a \int_0^d f(s, t) dt ds}_{\text{Summed-Area Table}} + \underbrace{\int_0^a \int_0^c f(s, t) dt ds}_{\text{Summed-Area Table}}\end{aligned}$$

Precomputing the 2D function:

$$S_{(0,0)}(x, y) \equiv \int_0^x \int_0^y f(s, t) dt ds$$

lets us evaluate integrals with four look-ups:

$$\int_a^b \int_c^d f(s, t) dt ds = S_{(0,0)}(b, d) - S_{(0,0)}(b, c) - S_{(0,0)}(a, d) + S_{(0,0)}(a, c)$$

over the rectangle $[a, b] \times [c, d]$ as:

$$\begin{aligned} \int_a^b \int_c^d f(s, t) dt ds &= \int_a^b \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds \\ &= \int_0^b \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds - \int_0^a \left(\int_0^d f(s, t) dt - \int_0^c f(s, t) dt \right) ds \\ &= \underbrace{\int_0^b \int_0^d f(s, t) dt ds}_{\text{look-up 1}} - \underbrace{\int_0^b \int_0^c f(s, t) dt ds}_{\text{look-up 2}} - \underbrace{\int_0^a \int_0^d f(s, t) dt ds}_{\text{look-up 3}} + \underbrace{\int_0^a \int_0^c f(s, t) dt ds}_{\text{look-up 4}} \end{aligned}$$

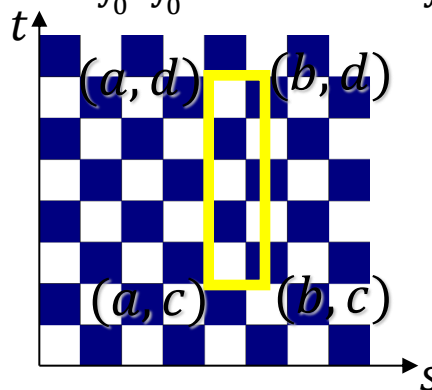


Summed-Area Tables (2D)

Integration (Pictorially):

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\boxed{\int_a^b \int_c^d f(s, t) dt ds} = \int_0^b \int_0^d f(s, t) dt ds - \int_0^b \int_0^c f(s, t) dt ds - \int_0^a \int_0^d f(s, t) dt ds + \int_0^a \int_0^c f(s, t) dt ds$$





Summed-Area Tables (2D)

Integration (Pictorially):

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\int_a^b \int_c^d f(s, t) dt ds = \boxed{\int_0^b \int_0^d f(s, t) dt ds} - \int_0^b \int_0^c f(s, t) dt ds - \int_0^a \int_0^d f(s, t) dt ds + \int_0^a \int_0^c f(s, t) dt ds$$

The diagram illustrates the integration process on a 2D grid. The grid is bounded by $(0,0)$ at the bottom-left and (b,d) at the top-right. The horizontal axis is labeled s and the vertical axis is labeled t . A yellow rectangle highlights the entire grid area. An orange rectangle highlights a smaller sub-region within the yellow rectangle.

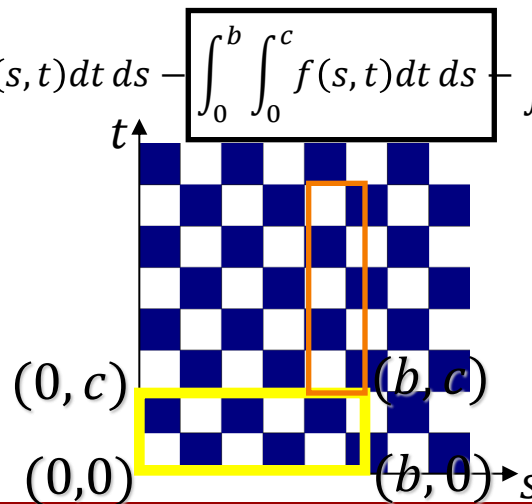


Summed-Area Tables (2D)

Integration (Pictorially):

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\int_a^b \int_c^d f(s, t) dt ds = \int_0^b \int_0^d f(s, t) dt ds - \int_0^b \int_0^c f(s, t) dt ds - \int_0^a \int_0^d f(s, t) dt ds + \int_0^a \int_0^c f(s, t) dt ds$$



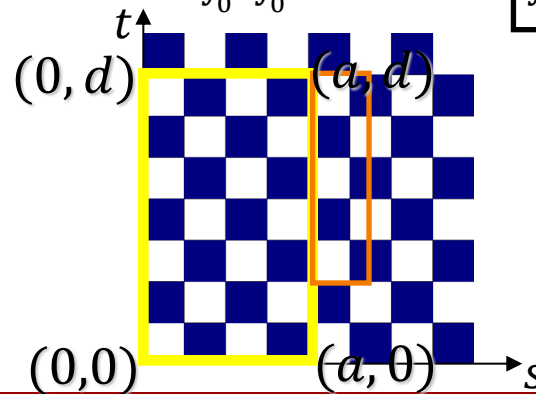


Summed-Area Tables (2D)

Integration (Pictorially):

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\int_a^b \int_c^d f(s, t) dt ds = \int_0^b \int_0^d f(s, t) dt ds - \int_0^b \int_0^c f(s, t) dt ds - \boxed{\int_0^a \int_0^d f(s, t) dt ds} + \int_0^a \int_0^c f(s, t) dt ds$$



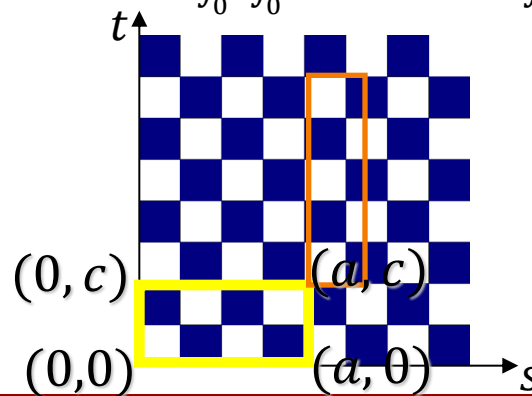


Summed-Area Tables (2D)

Integration (Pictorially):

In 2D, we can write out the integral of the function f over the rectangle $[a, b] \times [c, d]$ as:

$$\int_a^b \int_c^d f(s, t) dt ds = \int_0^b \int_0^d f(s, t) dt ds - \int_0^b \int_0^c f(s, t) dt ds - \int_0^a \int_0^d f(s, t) dt ds + \boxed{\int_0^a \int_0^c f(s, t) dt ds}$$





Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table



*Assuming the origin is at the bottom left corner



Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table

1			



*Assuming the origin is at the bottom left corner



Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table

1	3		



*Assuming the origin is at the bottom left corner



Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table

1	3	4	



*Assuming the origin is at the bottom left corner



Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table

1	3	4	7



*Assuming the origin is at the bottom left corner



Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table

5			
1	3	4	7



*Assuming the origin is at the bottom left corner



Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table

5	9		
1	3	4	7



*Assuming the origin is at the bottom left corner



Summed-Area Tables (Pre-Process)

Precompute the values of the integral:

$$S_{(0,0)}(a,b) = \int_0^a \int_0^b f(s,t) dt ds$$

Each summed-area table texel is the sum of all input texels below and to the left*

Input image

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Summed area table

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7



*Assuming the origin is at the bottom left corner

Summed-Area Tables (Run-Time)



Example:

Compute the average in the rectangle $[1,3] \times [2,3]$.
⇒ Compute the sum and divide by the area

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Input image

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7

Summed-area table



Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$.

$\Rightarrow$ Compute the sum and divide by the area

$$\text{Sum}([1,3] \times [2,3]) = S_{(0,0)}(3,3)$$

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Input image

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7

Summed-area table



Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$.

$\Rightarrow$ Compute the sum and divide by the area

$$\text{Sum}([1,3] \times [2,3]) = S_{(0,0)}(3,3) - S_{(0,0)}(0,3)$$

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Input image

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7

Summed-area table



Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$.

$\Rightarrow$ Compute the sum and divide by the area

$$\text{Sum}([1,3] \times [2,3]) = S_{(0,0)}(3,3) - S_{(0,0)}(0,3) - S_{(0,0)}(3,1)$$

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Input image

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7

Summed-area table



Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$.

$\Rightarrow$ Compute the sum and divide by the area

$$\text{Sum}([1,3] \times [2,3]) = S_{(0,0)}(3,3) - S_{(0,0)}(0,3) - S_{(0,0)}(3,1) + S_{(0,0)}(0,1)$$

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Input image

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7

Summed-area table



Summed-Area Tables (Run-Time)

Example:

Compute the average in the rectangle $[1,3] \times [2,3]$.

⇒ Compute the sum and divide by the area

$$\begin{aligned}\text{Sum}([1,3] \times [2,3]) &= S_{(0,0)}(3,3) - S_{(0,0)}(0,3) - S_{(0,0)}(3,1) + S_{(0,0)}(0,1) \\ &= 26 - 6 - 14 + 5 = 11\end{aligned}$$

$$\text{Average}([1,3] \times [2,3]) = \frac{\text{Sum}([1,3] \times [2,3])}{\text{Area}([1,3] \times [2,3])} = \frac{11}{6}$$

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Input image

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7

Summed-area table

Summed-Area Tables (Run-Time)



Precompute the values of the integral

- ✓ Constant time averaging, regardless of rectangle size
- ✗ If the input image has values in the range $[0,255]$ (i.e. one byte per channel), the summed area table can have values in the range $[0, 255 \cdot \text{width} \cdot \text{height}]$

1	2	4	0
0	3	1	1
4	2	0	1
1	2	1	3

Input image

6	15	21	26
5	12	14	19
5	9	10	14
1	3	4	7

Summed-area table



Overview

Texture mapping methods

- Parameterization
- Sampling

Texture mapping applications

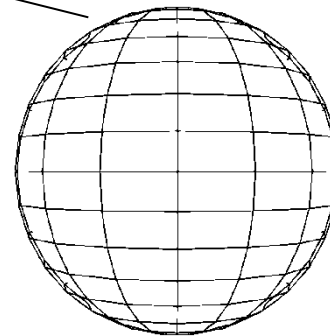
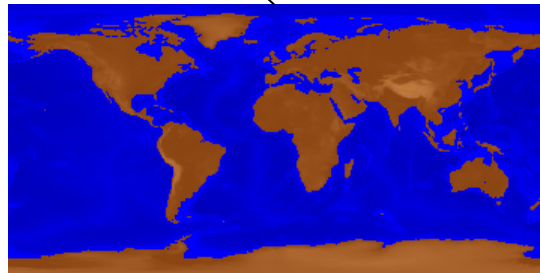
- Modulation textures
- Illumination mapping
- Bump mapping
- Environment mapping
- Shadow mapping



Modulation textures

Map texture values to **scale factor**

Modulation



$$I = \mathbf{T}(\mathbf{s}, \mathbf{t}) \left(I_E + K_A \cdot I_L^A + \sum_L (K_D \cdot \langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R}(\vec{L}) \rangle^{K_n}) \cdot I_L \right)$$

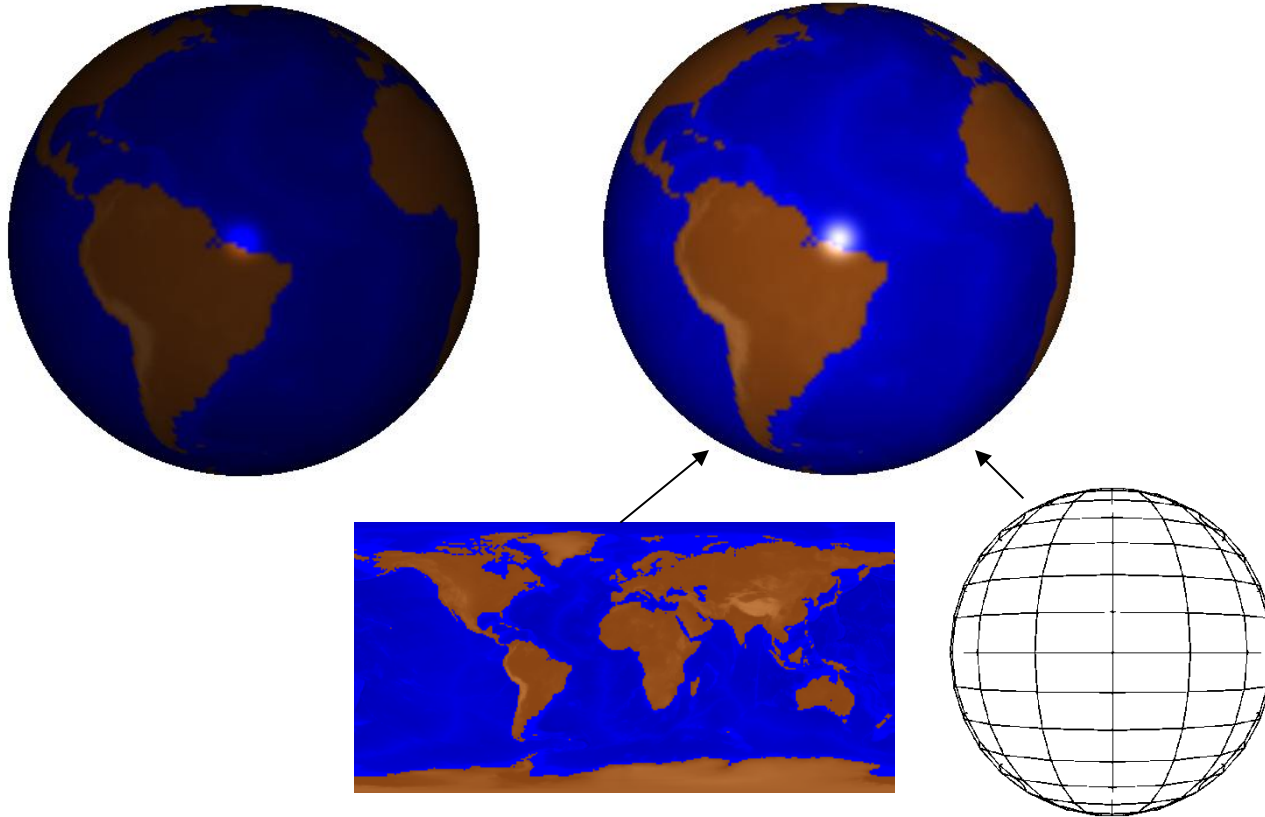


Illumination Mapping

Map texture values to a **material parameter**

Modulation

Diffuse



$$I = I_E + K_A \cdot I_L^A + \sum_L (T(s, t) \langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R}(\vec{L}) \rangle^{K_n}) \cdot I_L$$



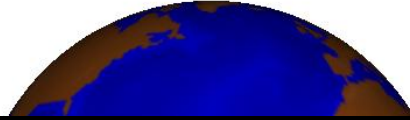
Illumination Mapping

Map texture values to a **material parameter**

Modulation

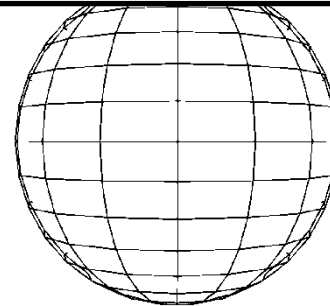
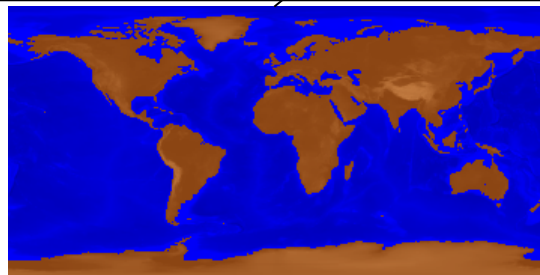


Diffuse



We evaluate the texture at each pixel but can use the interpolated lighting values $\langle \vec{N}, \vec{L} \rangle$ from the corners

This requires the graphics card to separately store the diffuse component of the lighting at each vertex

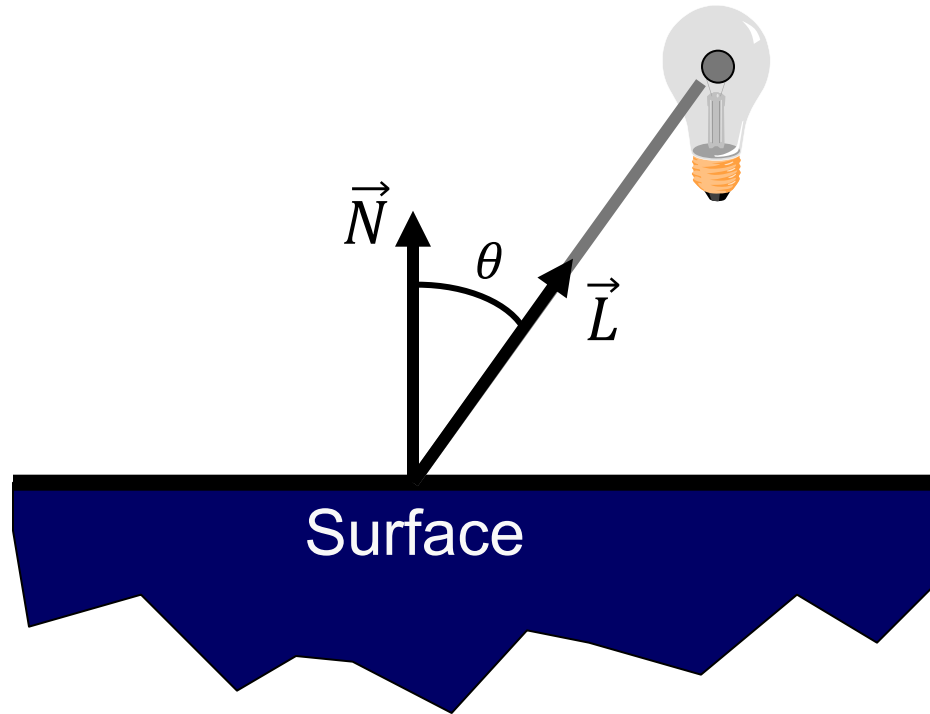


$$I = I_E + K_A \cdot I_L^A + \sum_L (T(s, t) \langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R}(\vec{L}) \rangle^{K_n}) \cdot I_L$$



Bump Mapping

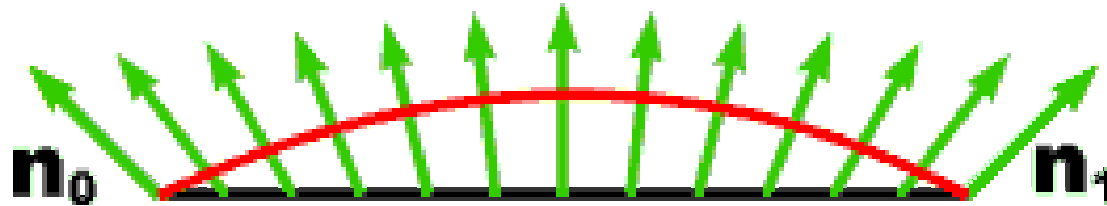
Recall that many parts of our lighting calculation depend on surface normals



$$I = I_E + K_A \cdot I_L^A + \sum_L (\langle \vec{N}, \vec{L} \rangle + K_S \cdot \langle \vec{V}, \vec{R}(\vec{L}) \rangle^{K_n}) \cdot I_L$$



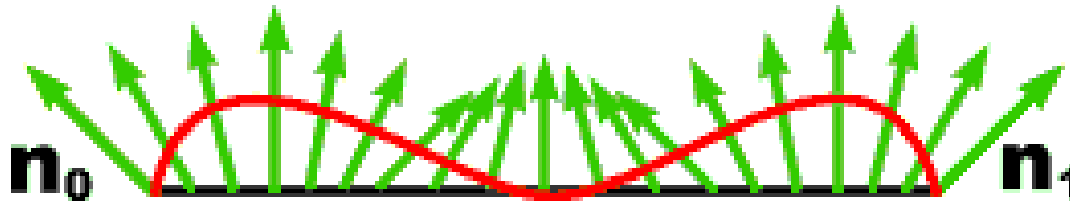
Bump Mapping



Phong shading performs per-pixel lighting calculations with the interpolated normal



approximates a smoothly curved surface

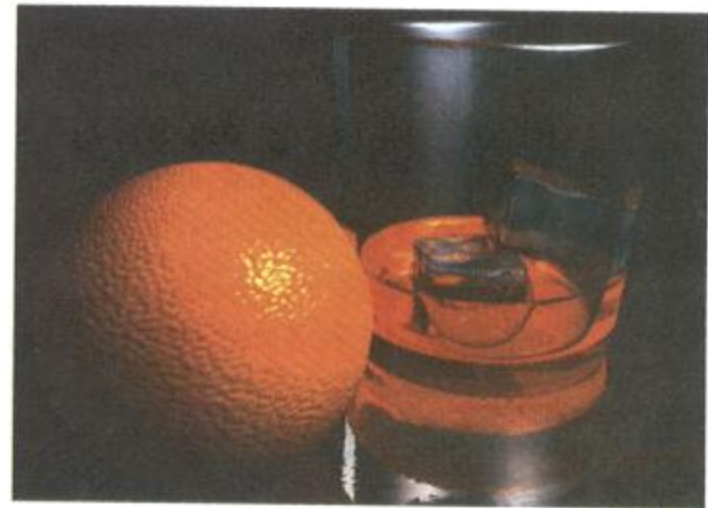


Bump maps encode the normals in the texture



approximates a more complex undulating surface

Bump Mapping



Bump mapping does not change object silhouette

Environment Mapping



Goal:

Render shiny surfaces so they the reflect the world.

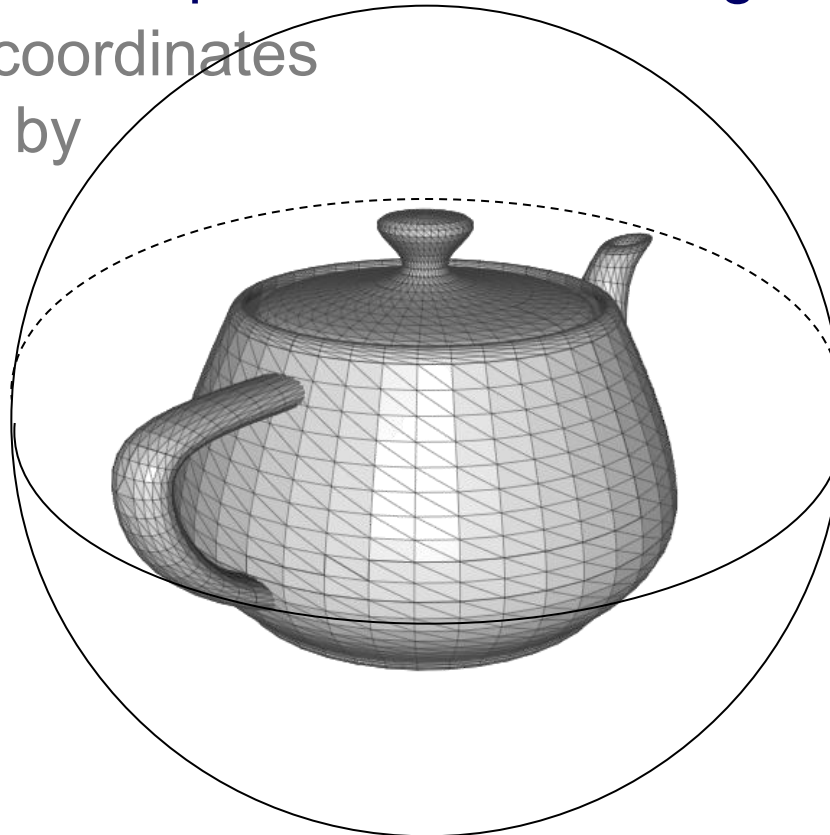


Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

- Pre-compute a map of the surrounding environment
- Set texture coordinates dynamically by reflecting



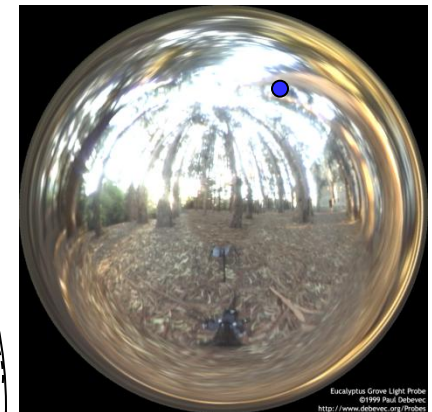
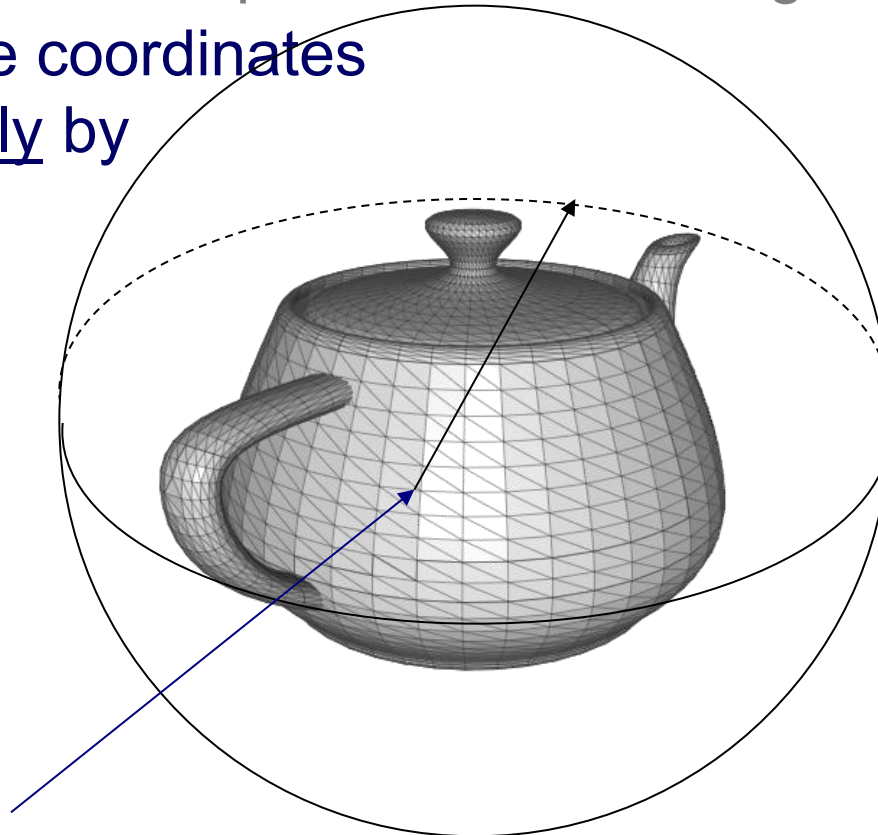


Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

- Pre-compute a map of the surrounding environment
- Set texture coordinates dynamically by reflecting



Eucalyptus Grove Light Probe
©1999 Paul Debevec
<http://www.debevec.org/Probes>

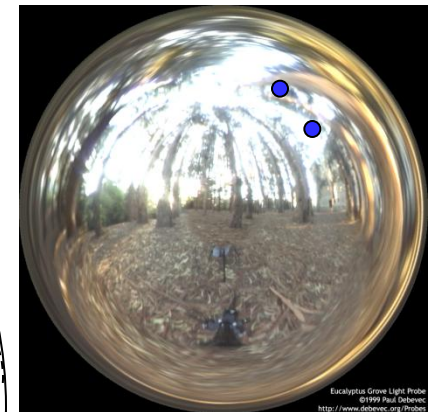
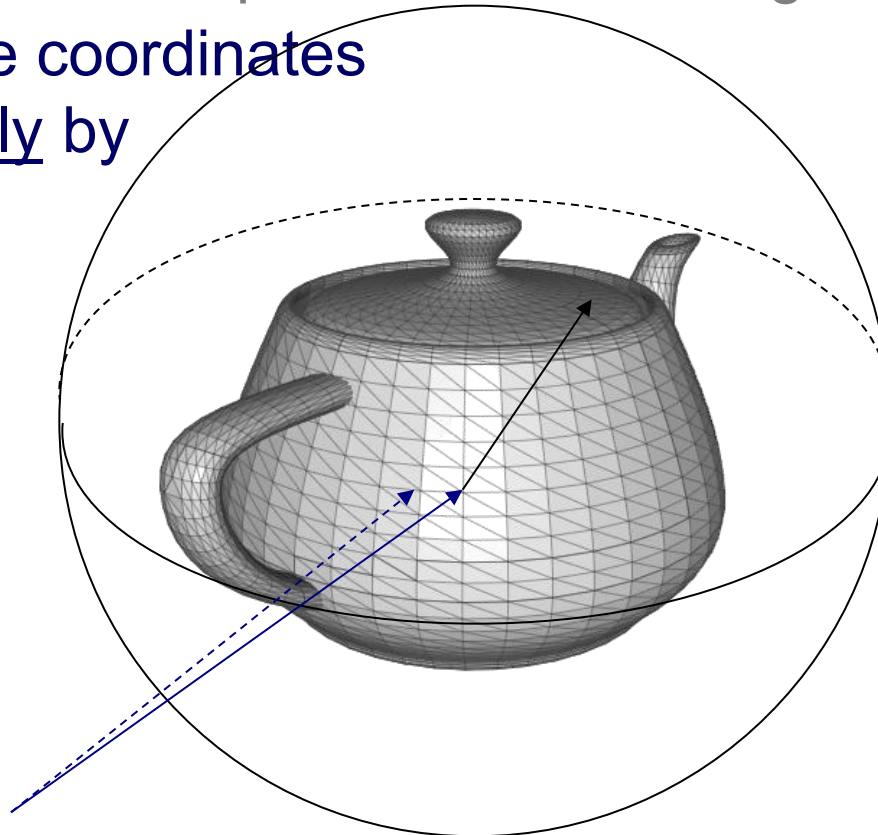


Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

- Pre-compute a map of the surrounding environment
- Set texture coordinates dynamically by reflecting



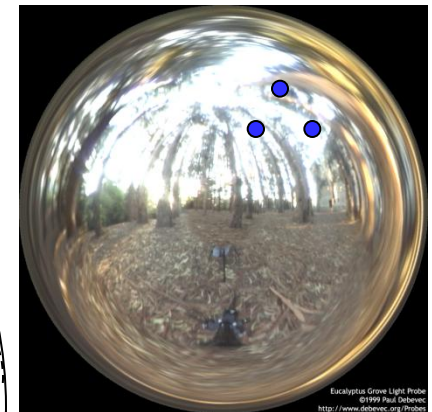
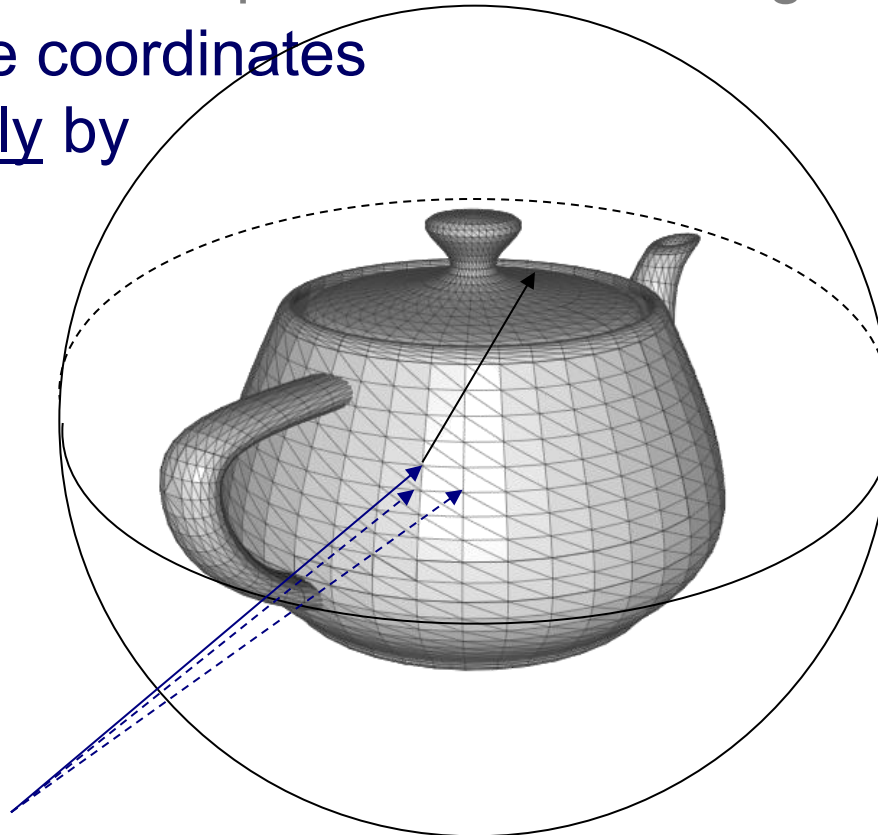


Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

- Pre-compute a map of the surrounding environment
- Set texture coordinates dynamically by reflecting



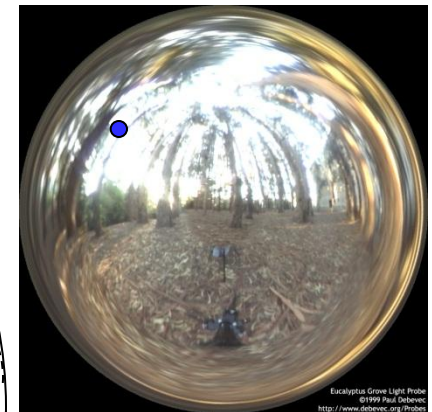
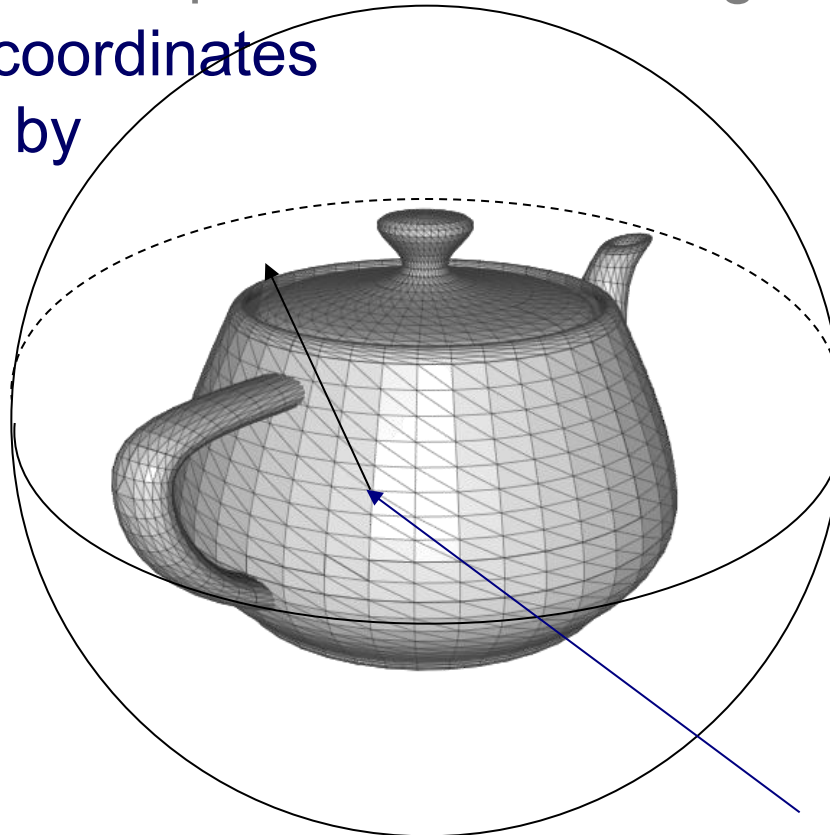


Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

- Pre-compute a map of the surrounding environment
- Set texture coordinates dynamically by reflecting



For the same triangle, changing the position of the camera changes the texture coordinates.

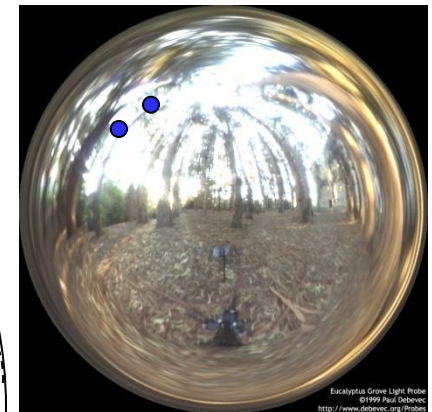
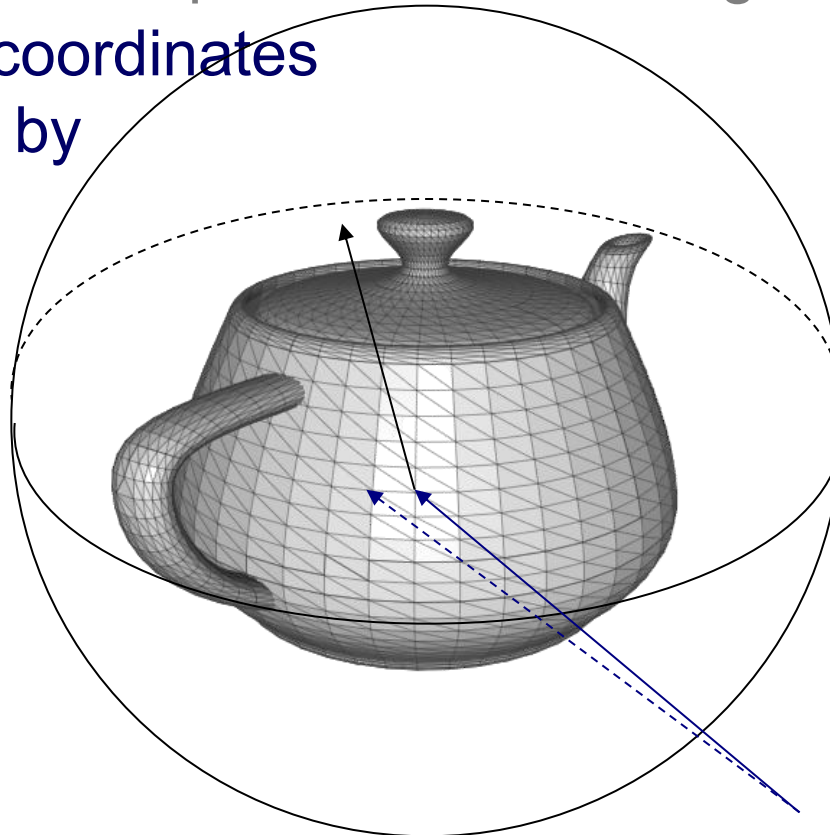


Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

- Pre-compute a map of the surrounding environment
- Set texture coordinates dynamically by reflecting



For the same triangle, changing the position of the camera changes the texture coordinates.

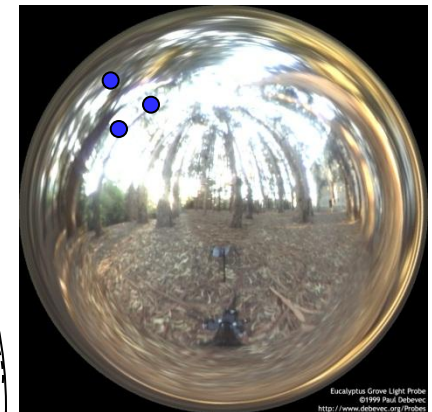
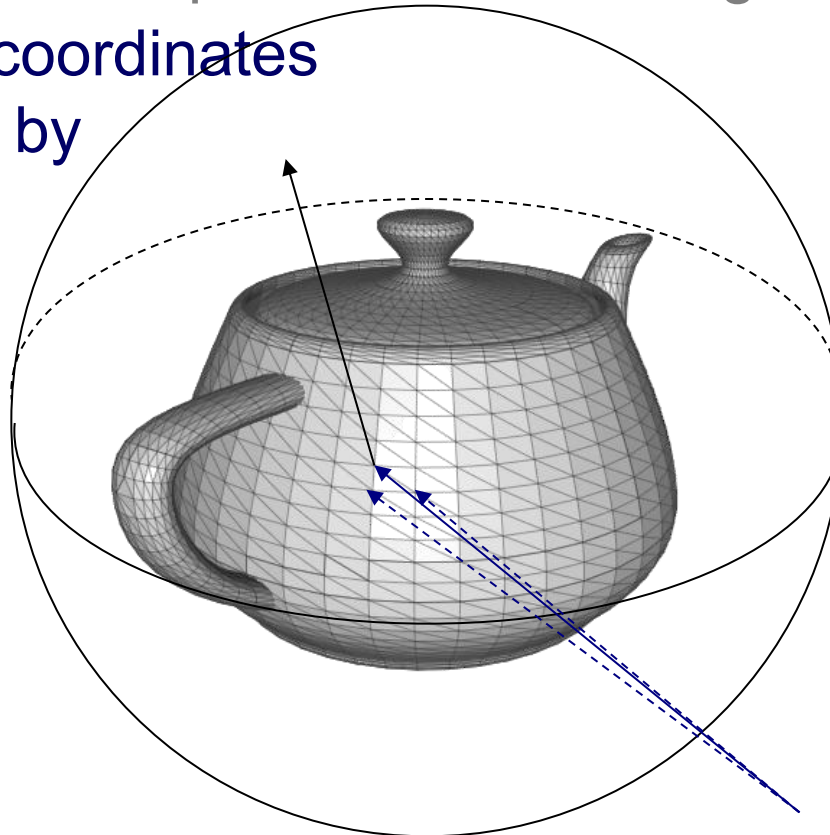


Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

- Pre-compute a map of the surrounding environment
- Set texture coordinates dynamically by reflecting



For the same triangle, changing the position of the camera changes the texture coordinates.



Environment Mapping

Goal:

Render shiny surfaces so they reflect the world.

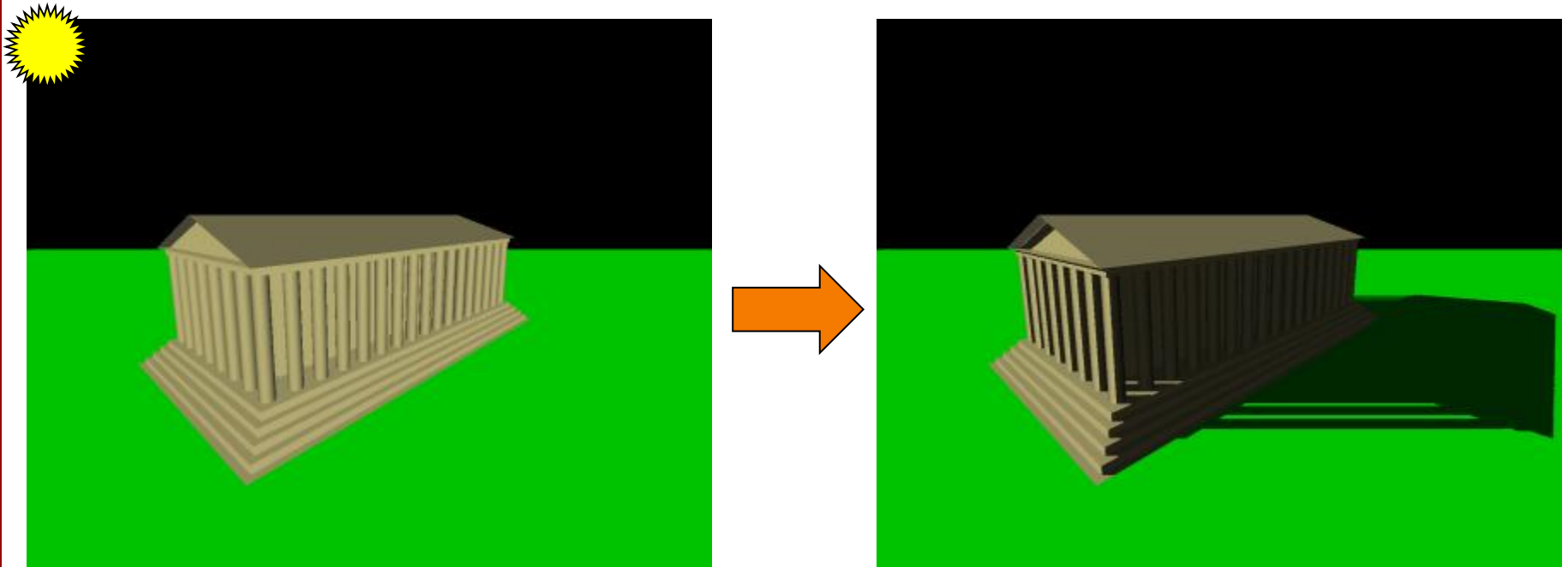


P. Debevec

Shadow Mapping (Williams 1978)



Test if surface is in shadow when computing the contribution to the lighting equation.



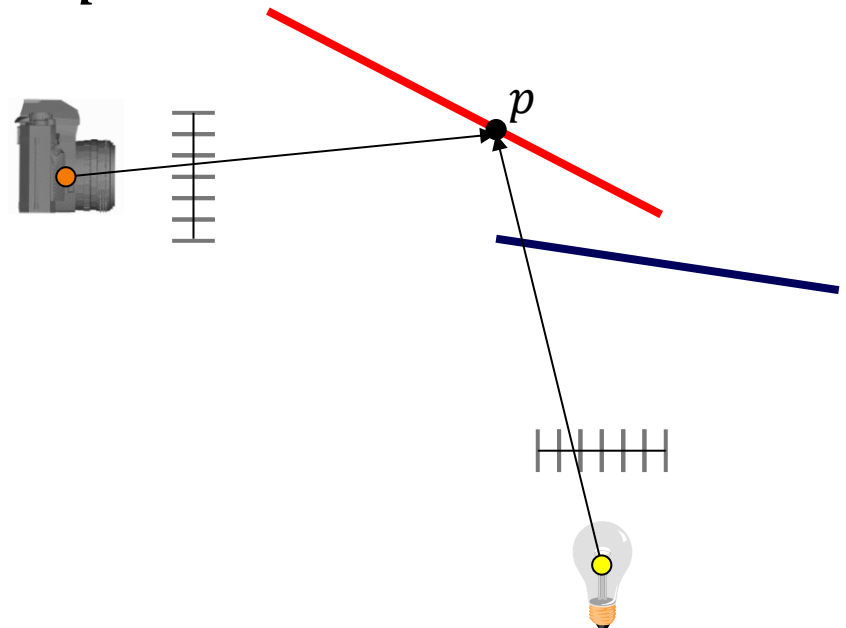
Images courtesy of https://en.wikipedia.org/wiki/Shadow_mapping

Shadow Mapping (Williams 1978)



Q: Is a point p , seen by the camera, in shadow with respect to the light?

A: The point is visible to the light if **rendering from the light's perspective**, p 's z -coordinate is the value in the z -buffer.

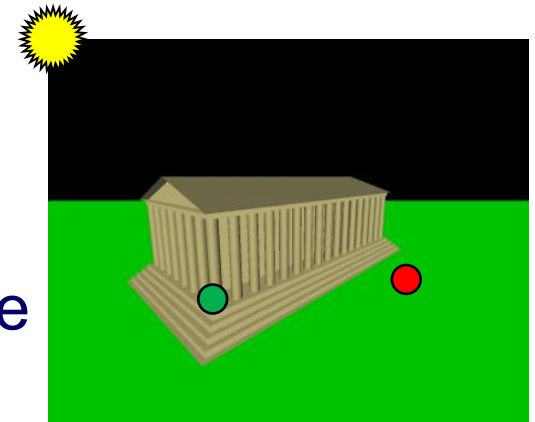


Shadow Mapping (Williams 1978)

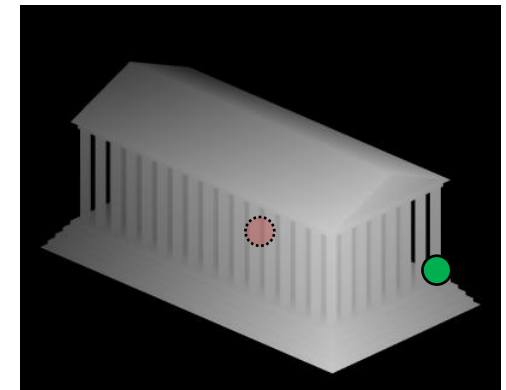


Algorithm:

- **Render the scene from the light's perspective** and read back the z-buffer/shadow map.
- For each camera view pixel, compute its z-coordinate relative to the light



Camera view



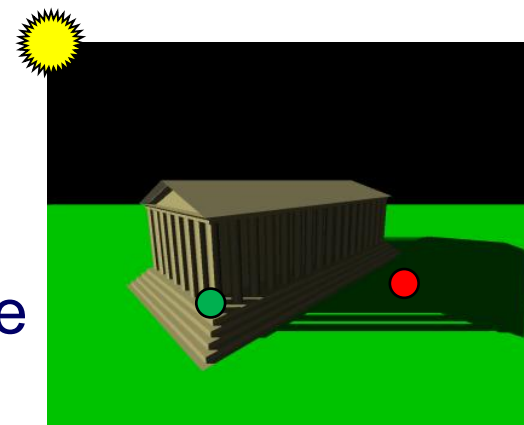
Shadow map

Shadow Mapping (Williams 1978)

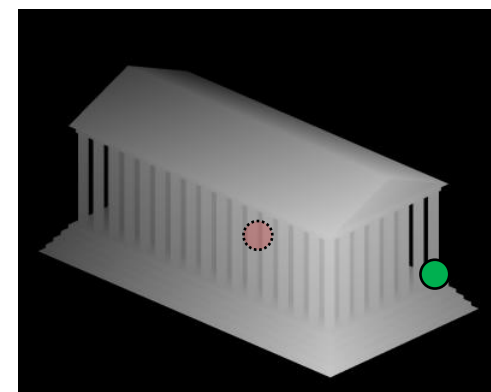


Algorithm:

- **Render the scene from the light's perspective** and read back the z-buffer/shadow map.
- For each camera view pixel, compute its z-coordinate relative to the light
 - » If it's further back than the value in the shadow map, it's in shadow
 - » Otherwise, it's illuminated



Camera view



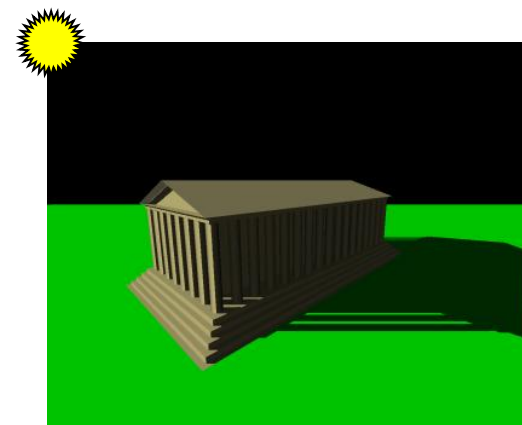
Shadow map

Shadow Mapping (Williams 1978)

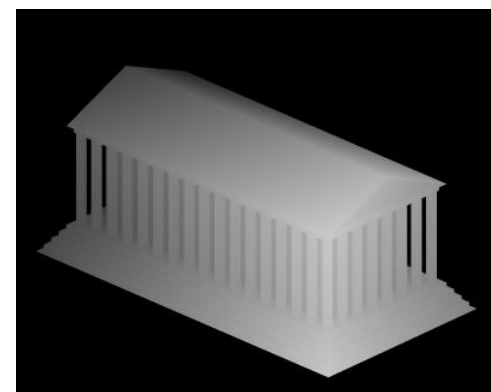


Note:

- The projection used for rendering from the light-source depends on the type of light:
 - » Directional → Parallel
 - » Point → Perspective
- Need to use multiple shadow maps if there are multiple lights in the scene



Camera view



Shadow map