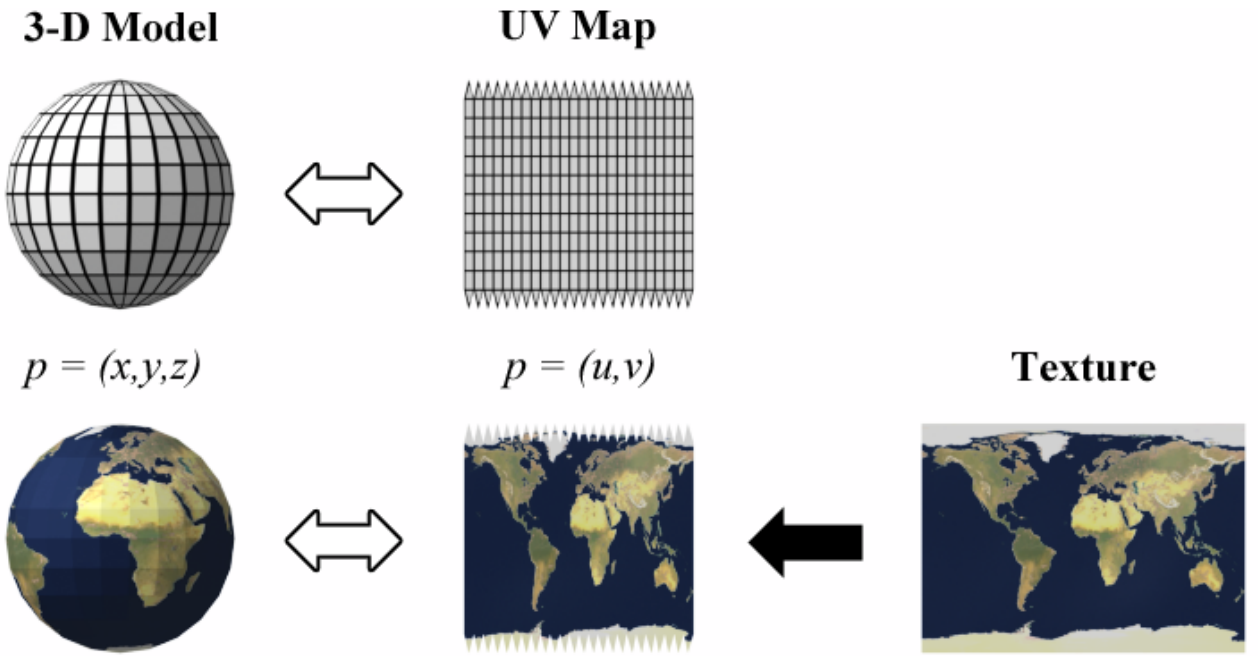
CS475/CS675 Computer Graphics

Texture

Texture

- Add surface detail
- Paste a photograph over a surface to provide detail.
- Texture can change surface colour or modulate surface colour.



http://en.wikipedia.org/wiki/UV_Mapping

Texture Mapping

Why use texture mapping to increase detail?

http://en.wikipedia.org/wiki/Microsoft_Flight_Simulator

Expensive solution: add more detail to model

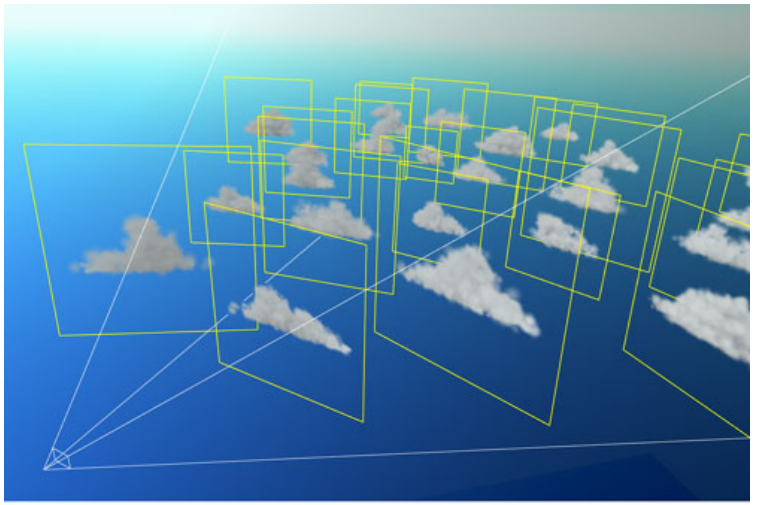
- \+ detail incorporated as a part of object
- – modeling tools aren't very good for adding detail
- – model takes longer to render
- – model takes up more space in memory
- – complex detail cannot be reused

Efficient solution: map a texture onto model

- \+ texture maps can be reused
- \+ texture maps take up space in memory, but can be shared, and compression and caching techniques can reduce overhead significantly compared to real detail
- \+ texture mapping can be done quickly (we'll see how)
- \+ placement and creation of texture maps can be made intuitive (e.g., tools for adjusting mapping, painting directly onto object)
- – texture maps do not affect the geometry of the object

Texture Mapping

- What kind of detail can go into these maps?
  - diffuse, ambient and specular colors
  - specular exponents
  - transparency, reflectivity
  - fine detail surface normals (bumps)
  - data to visualize
  - projected lighting and shadows
  - games use “billboards” for distant detail.



Courtesy of Mark Harris, UNC-Chapel Hill

http://www.flipcode.com/archives/Light_Mapping_Theory_and_Implementation.shtml

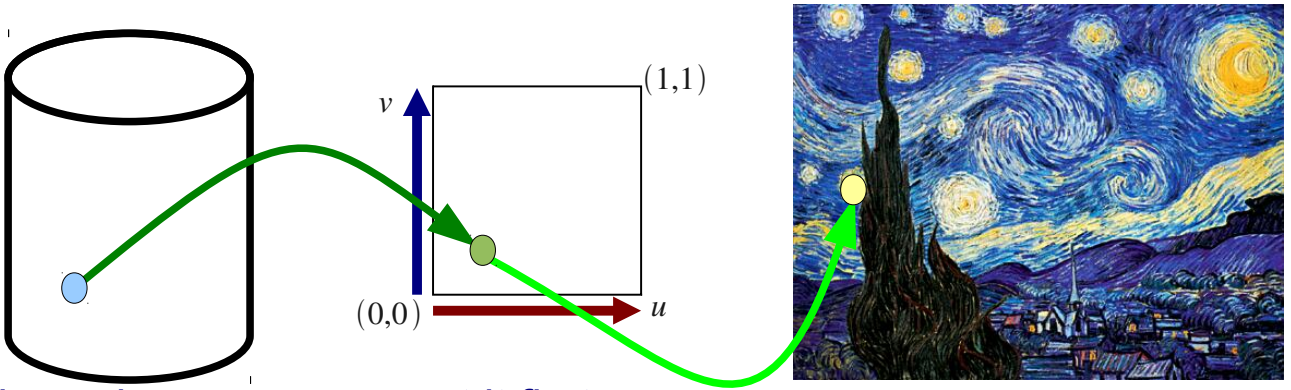
Texture Mapping

- What are Mappings?
  - A function is a mapping
    - functions map values in their subset of a domain into their subset of a co-domain
    - each value in the domain will be mapped to one value in the co-domain
  - Can transform one space into another with a function
    - Viewing pipeline takes the WCS to VCS to CCS to NDCS – these are all mappings.

- What is Texture Mapping?
  - We have points on a surface in object-space – the domain
  - We want to get values from a texture map – the co-domain
  - What function(s) should we use?

Texture Mapping

- **Basic Idea**
  - Texture mapping is the process of mapping a geometric point to a color in a texture map
  - We want to map arbitrary geometry to a pix(-cture)map of arbitrary dimension
  - We do this in two steps:
    - map a point on the arbitrary geometry to a point on an abstract unit square. (uv plane)
    - map a point on abstract unit square to a point on the actual pixmap of arbitrary dimension

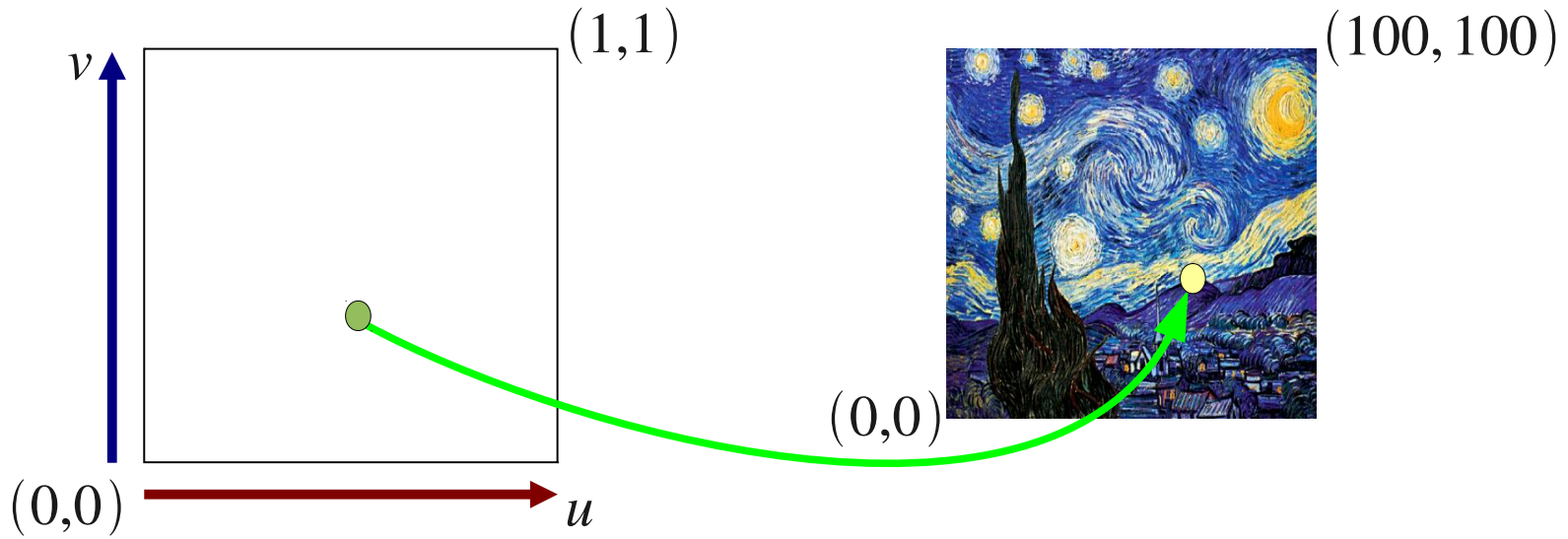


  - Second step is easier, so we present it first

Van Gogh

Texture Mapping

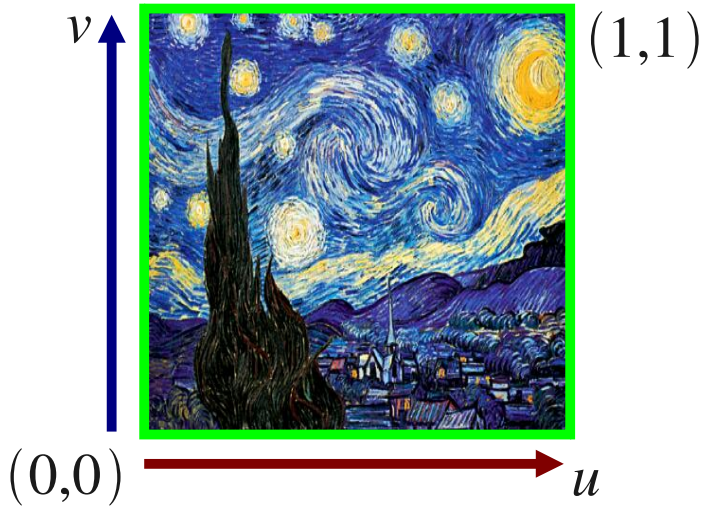
- **From a unit square to a pixmap**
  - A 2D example: mapping from unit *u,v* square to texture map (arbitrary pixmap, possibly with alpha values)



  - **Step 1**: transform a point on abstract continuous texture plane to a point in discrete texture map
  - **Step 2**: Get color at transformed point in texture image, for e.g., (0.0, 0.0) => (0, 0), (1.0, 1.0) => (100, 100), (0.75, 0.45) => (75, 45)

Texture Mapping

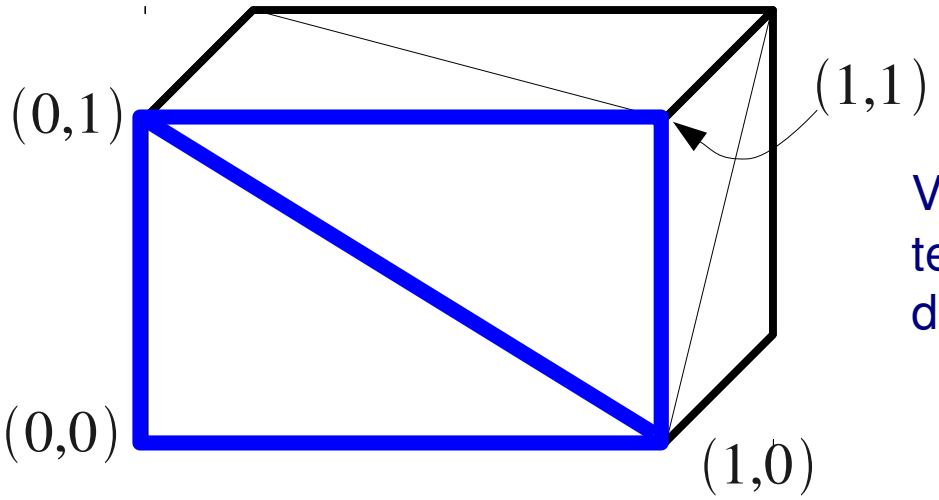
- In general, for any point *(u, v)* on unit plane, the corresponding point on the texture image is: *(u* * pixmap width, *v* * pixmap height).



- There are infinitely many points on unit plane so we will get *sampling errors*.
- *The uv* plane is continuous version of the texture pixmap.
- Now the unit *uv* square acts as stretchable rubber sheet to wrap around the texture mapped object.

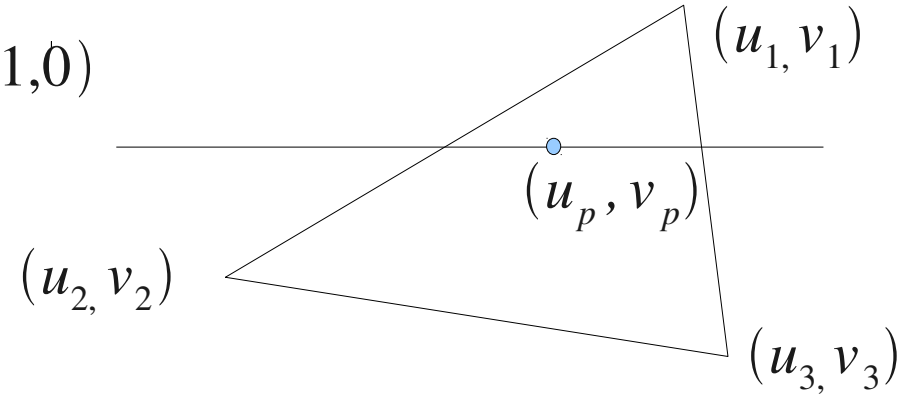
Texture Mapping Polygons

- **Interpolating Texture Coordinates**
  - Pre-calculate texture coordinates for each vertex, for e.g., as shown for the cuboid below.



Vertices will have different texture coordinates on different faces.

Interpolate *uv* coordinates linearly across triangles as part of Gouraud shading



Texture Mapping Polygons

- **Interpolating Texture Coordinates**
  - *Affine texture mapping* will interpolate texture coordinates linearly across the screen. This results in distorted textures.

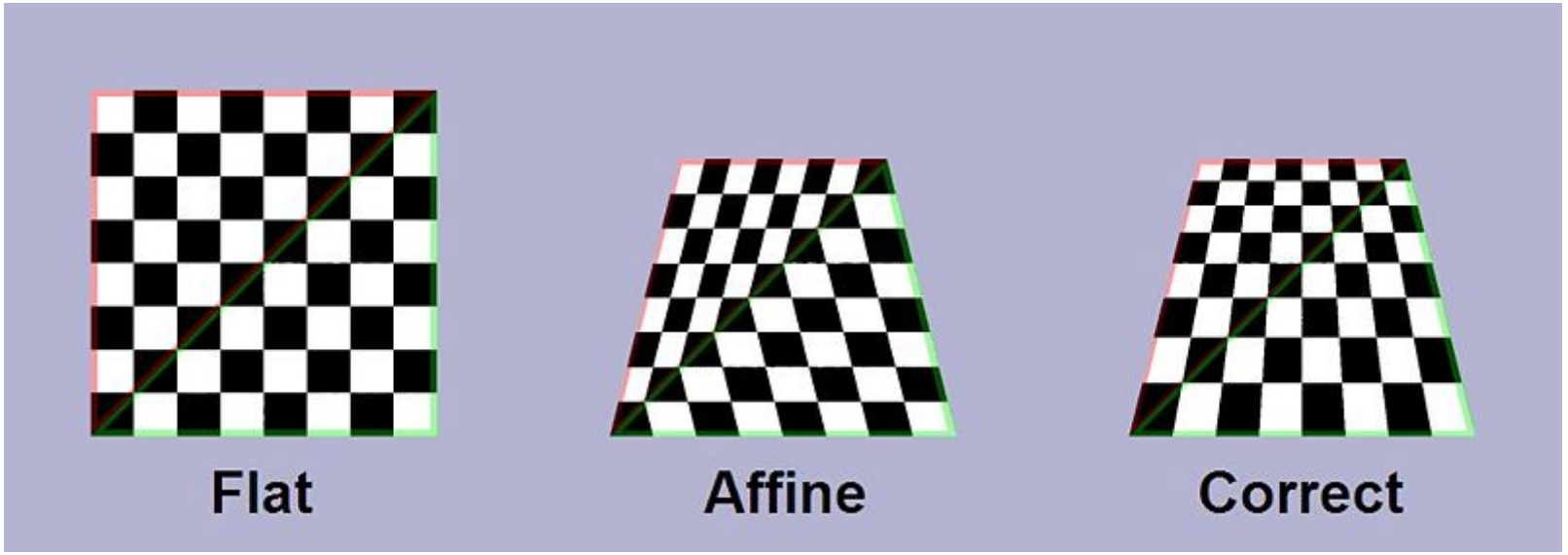
$$u_t = (1-t)u_0 + tu_1, 0 \le t \le 1$$

  - *Perspective texture mapping* accounts for the actual 3D positions of vertices.
    - This is expensive. Instead of interpolating the texture coordinates directly, the coordinates are divided by their depth (relative to the viewer), and the reciprocal of the depth value is also interpolated and used to recover the perspective-correct coordinate.

$$u_t = \frac{(1-t)\frac{u_0}{z_0} + t\frac{u_1}{z_1}}{(1-t)\frac{1}{z_0} + t\frac{1}{z_1}}, 0 \le t \le 1$$

Texture Mapping Polygons

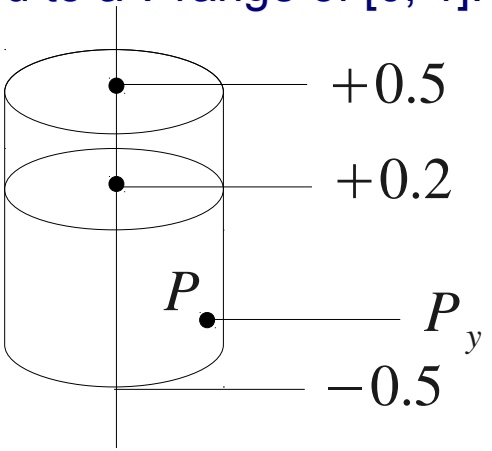
- **Interpolating Texture Coordinates**



http://en.wikipedia.org/wiki/Texture_mapping

Texture Mapping

- Cylinders and Cones
  - Imagine a standard cylinder or cone as a stack of circles. Use position of point on perimeter to determine *u,* use height of point in stack to determine *v. M*ap top and bottom caps separately, as onto a plane
  - Calculating *v* : height of point in object space, which ranges between [-.5, .5], gets mapped to a *v* range of [0, 1].

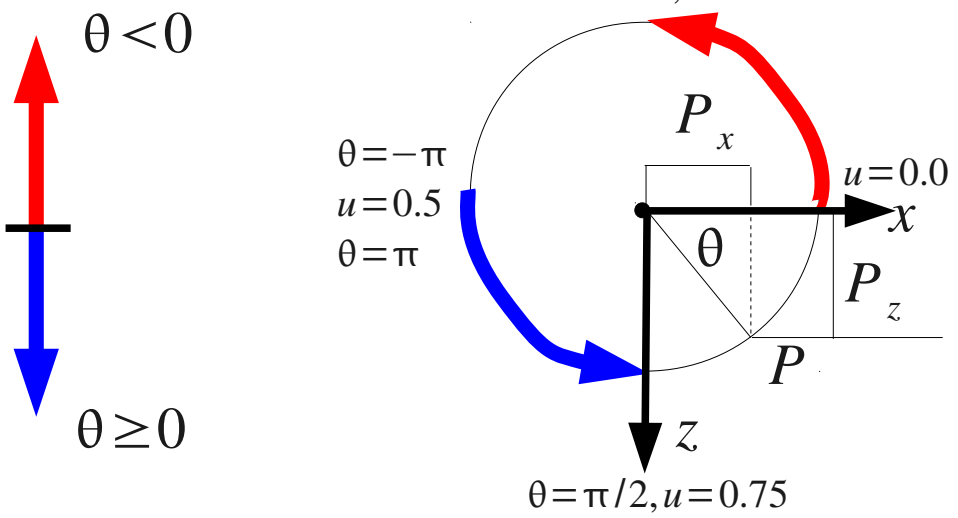


$$v = P_y + 0.5$$

  - Calculating *u :* map points on circular perimeter to *u* values between 0 and 1; 0 radians is 0, 2π radians is 1.
  - Then *u* = θ/2π. These mappings are arbitrary: any function mapping the angle around the cylinder (θ) to the range 0 to 1 will work.

Texture Mapping

- Cylinders and Cones
  - Want to convert a point P on the perimeter to an angle. θ is measured clockwise due to the right-handed coordinate system, but one still expects ***u*** to increase as we travel circle counter-clockwise, as shown by colored arrows below.



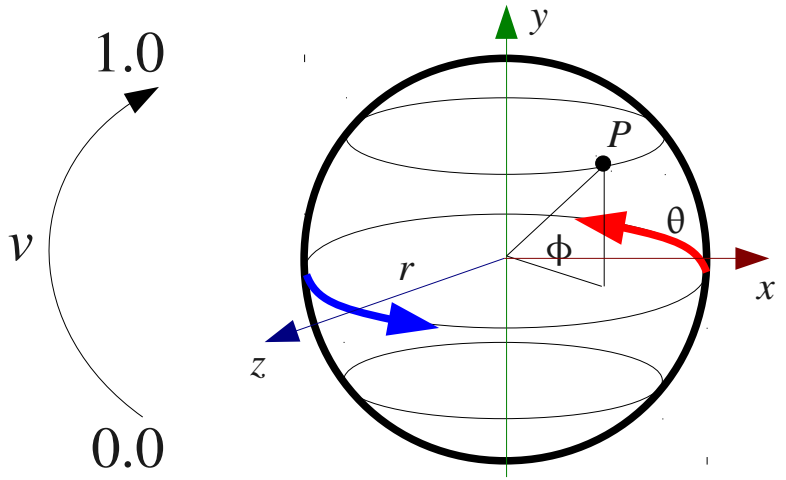
$$\theta = atan2(P_z, P_x)$$

$$if\ \theta < 0, u = -\theta/2\pi \quad [0.0, 0.5]$$
$$if\ \theta \geq 0, u = 1 - \theta/2\pi \quad [0.5, 1.0]$$

  - Circle is not necessarily a unit circle
  - so use arctangent instead of sine or cosine
  - Use atan2 to get whole circle (atan gives only half a circle).

Texture Mapping

- Sphere
  - A sphere is a stack of circles of varying radii.
  - Calculating *u* : as explained previously.



$$\phi = \sin^{-1}\left(\frac{P_y}{r}\right), -\frac{\pi}{2} \le \phi < \frac{\pi}{2}$$

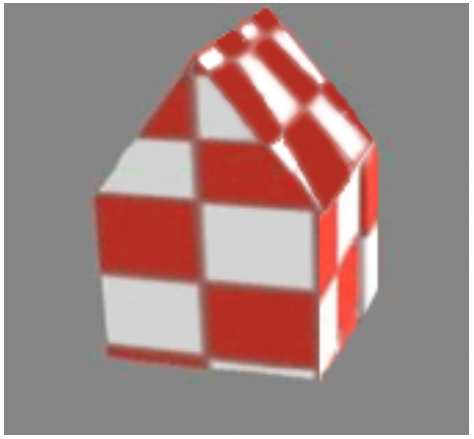
$$v = \frac{\phi}{\pi} + 0.5$$

  - Calculating *v : v* is the latitude and varies from 0 to 1.
    - When *v* is 0 or 1, there is a singularity and *u* should equal so specific value.
    - Do not specifically check for this because we will rarely see this value within floating point precision.

Texture Mapping

- Complex Geometries
  - Texture mapping of simple polygons was easy.
  - How do we texture map more complicated shapes? E.g. A house
  - How should we texture map it?
  - We could texture map each polygonal face of the house

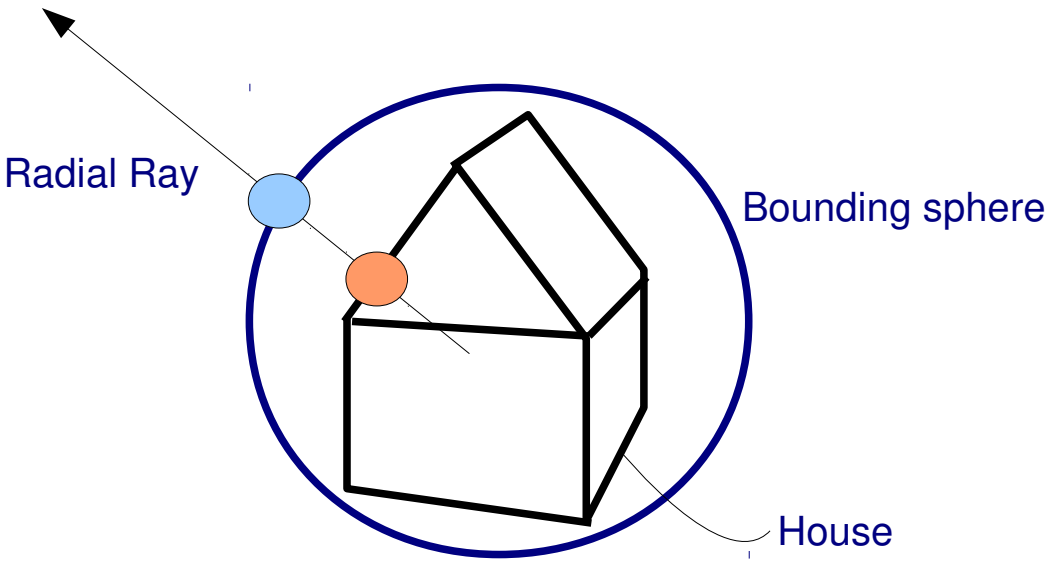
    (we know how to do this already as they are planar).



  - This causes discontinuities at edges of polygons. We want smooth mapping without edge discontinuities
  - Intuitive approach: reduce to a solved problem. Pretend the house is a sphere for texture-mapping purposes
  - Texture mapping in raytracing

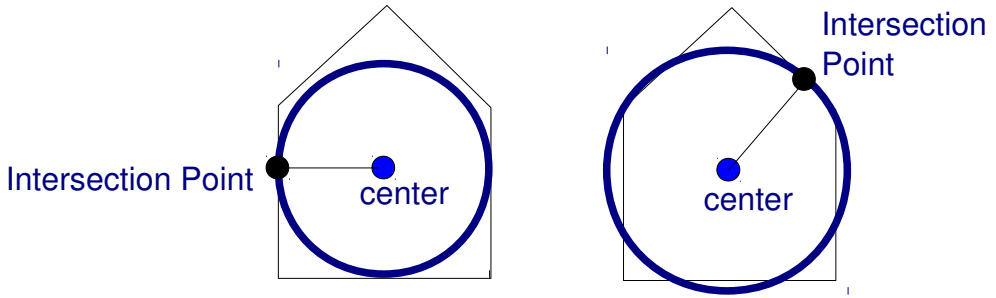
Texture Mapping

- Complex Geometries
  - Intuitive approach: place bounding sphere around complicated shape
  - *uv*-coords using Spherical mapping



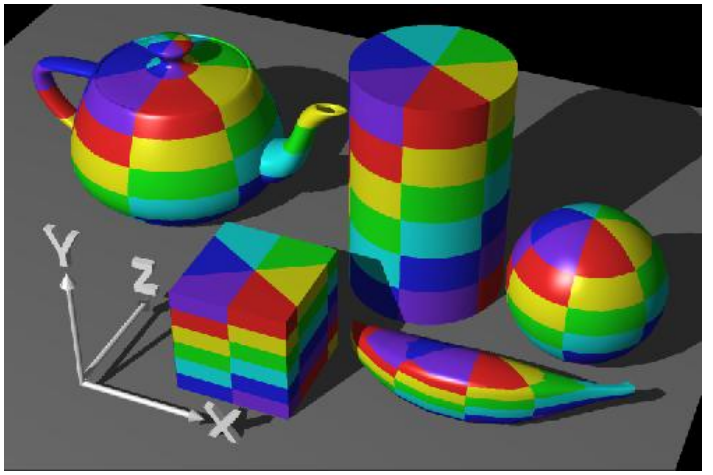
Texture Mapping

- Complex Geometries
  - Another Method using an inscribed sphere
  - Assume intersection point lies on some sphere and calculate its uv coordinates using spherical projection.
  - Use the distance from the center of the object to the intersection point as radius for the sphere.
    The sphere changes with the point's location.

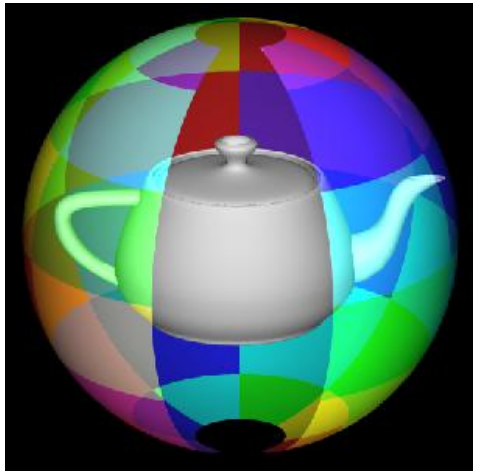


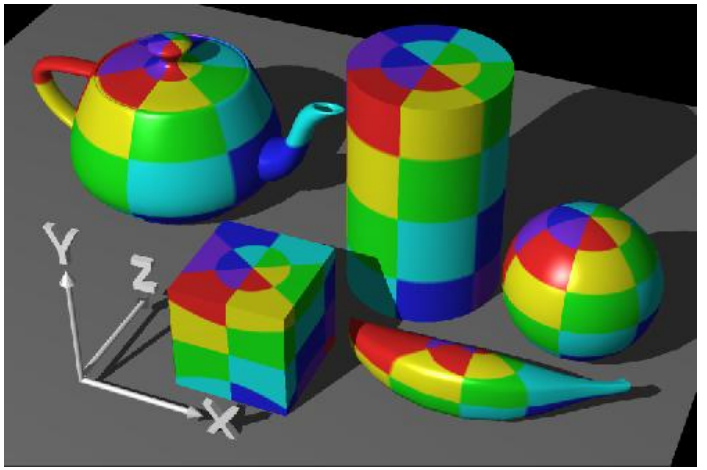
Texture Mapping

- Complex Geometries



Cylindrical Texture Mapping



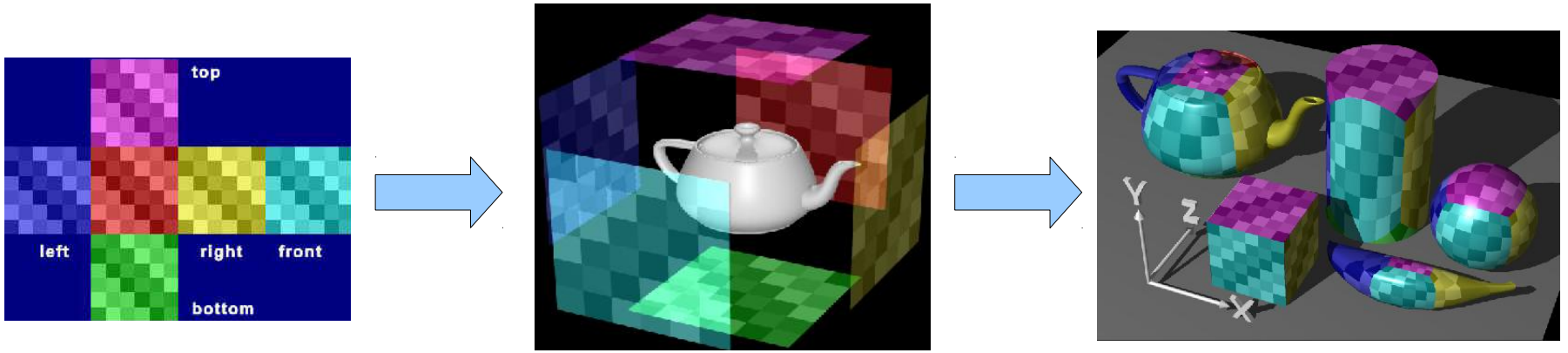


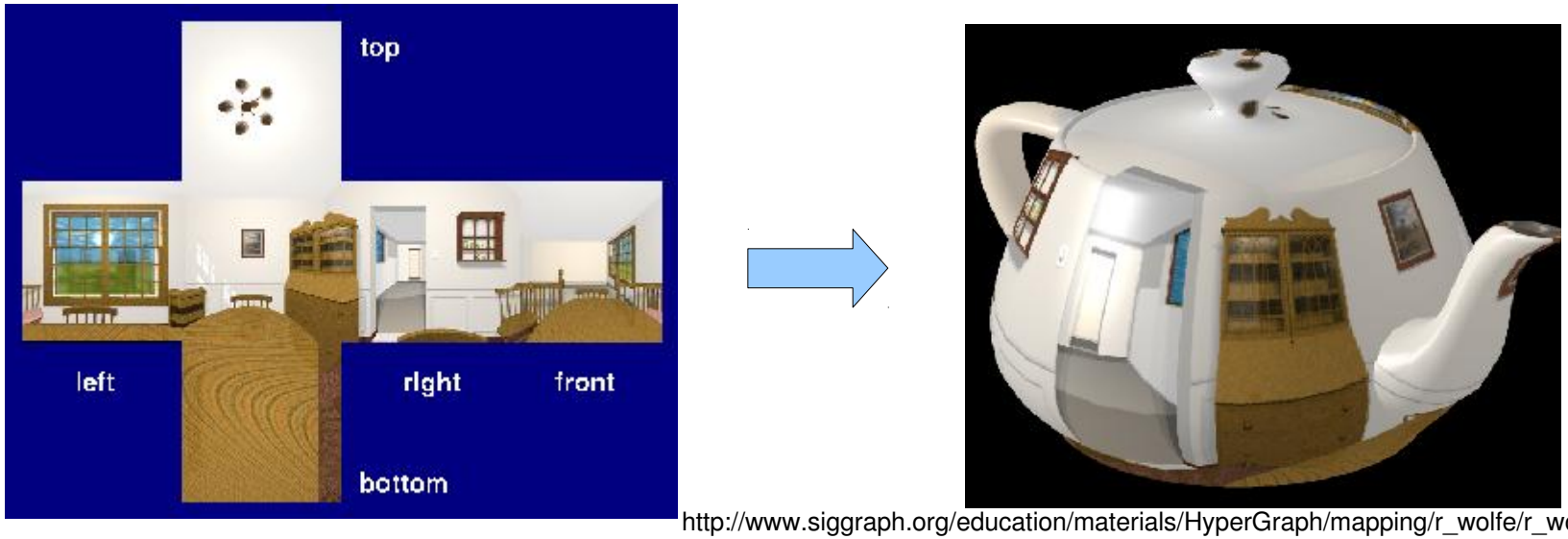
Spherical Texture Mapping

http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

Environment Mapping

- Faking reflections



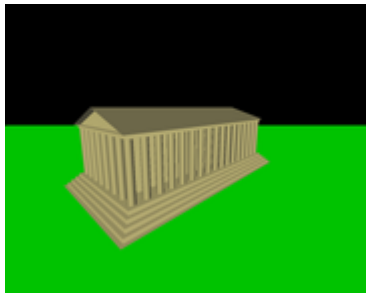
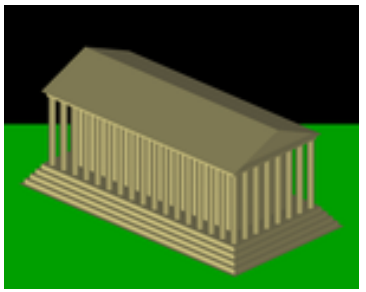
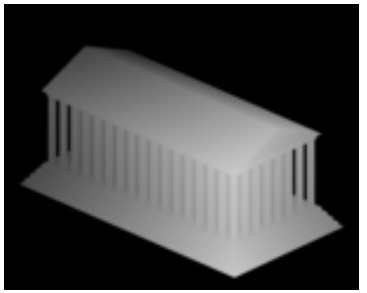
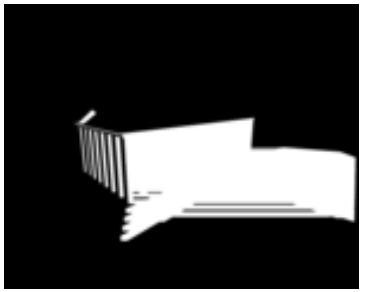
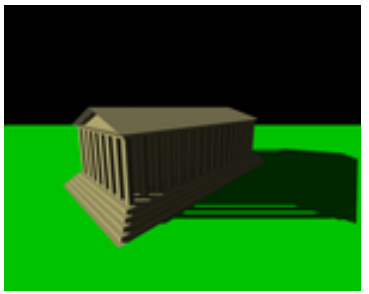


http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

Cube (environment) map is made by stitching together 6 ***projective*** textures.

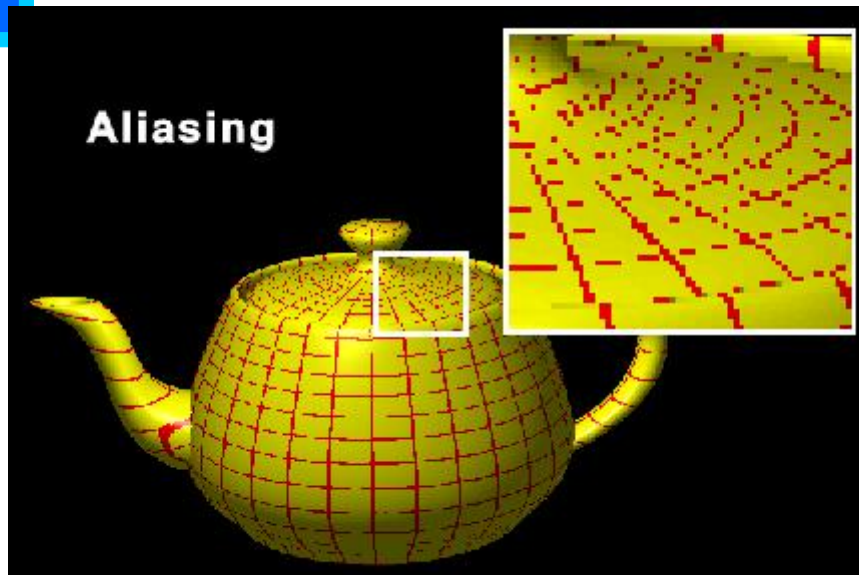
Shadow Mapping

- Faking shadows
- Transform camera to each directional light source
- Render the scene from light source point of view (keeping the same far clipping plane), only updating Z buffer
- Read Z buffer into a texture map (the shadow map)
- While rendering the scene from the original eye point, convert every pixel to light space.
- If in light space the distance from the pixel to the light is greater than value in shadow map, it is in shadow.
- Major aliasing problem because size of shadow maps is limited.
- Implemented on hardware

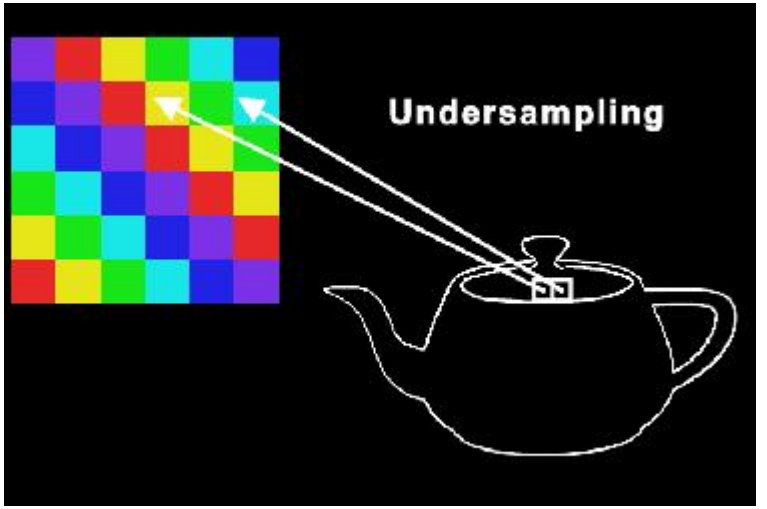






http://en.wikipedia.org/wiki/Shadow_mapping

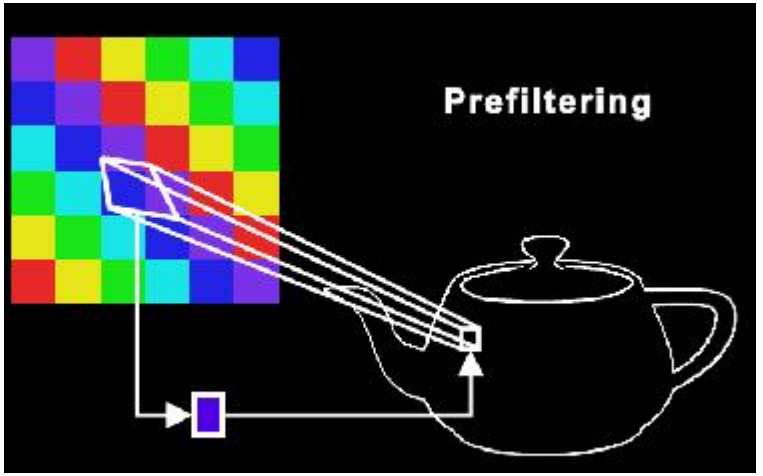
Texture Aliasing



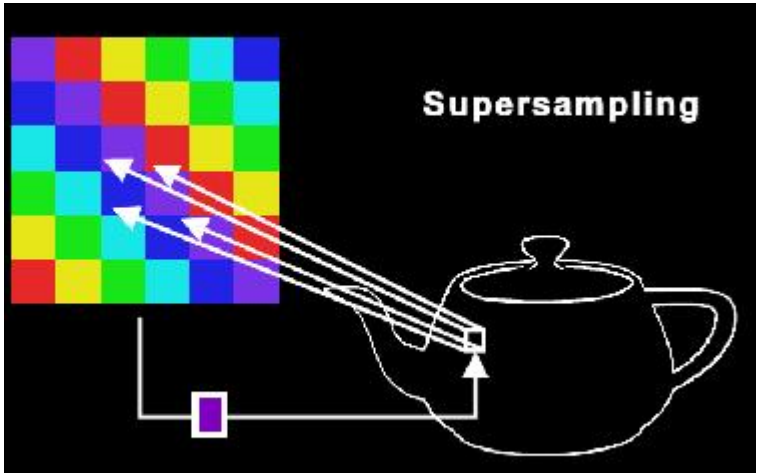
Reason



Solution 1

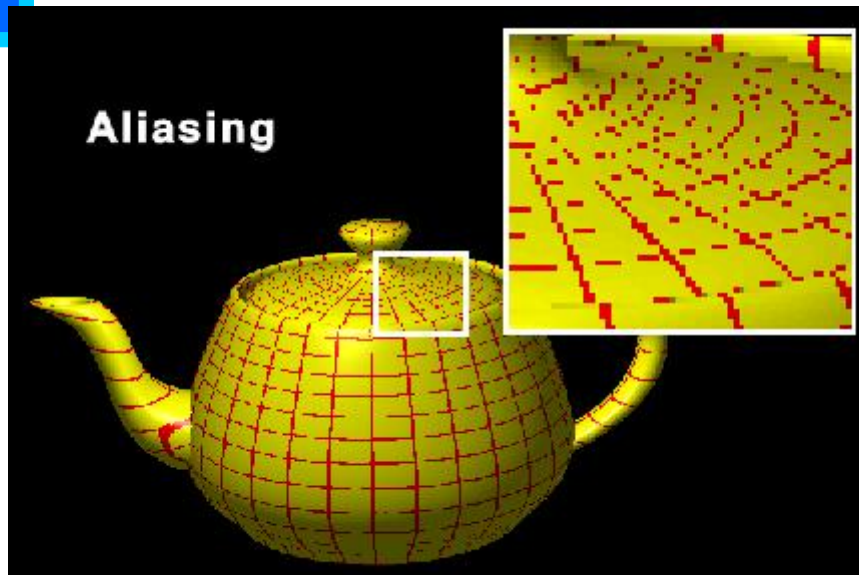


Solution 2

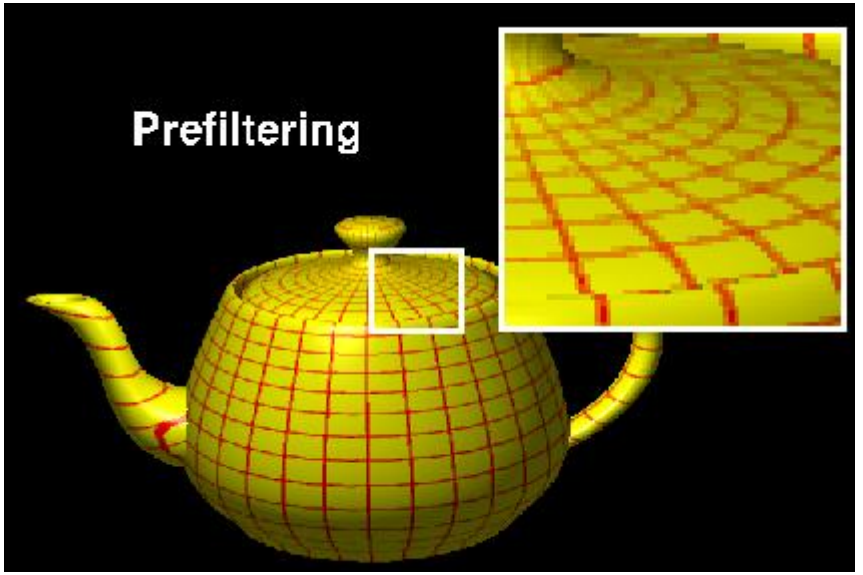


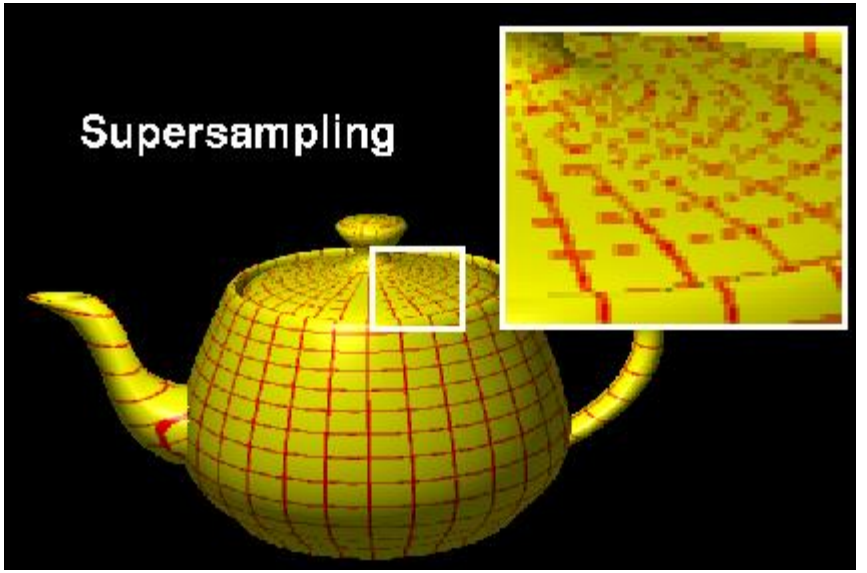
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

Texture Aliasing



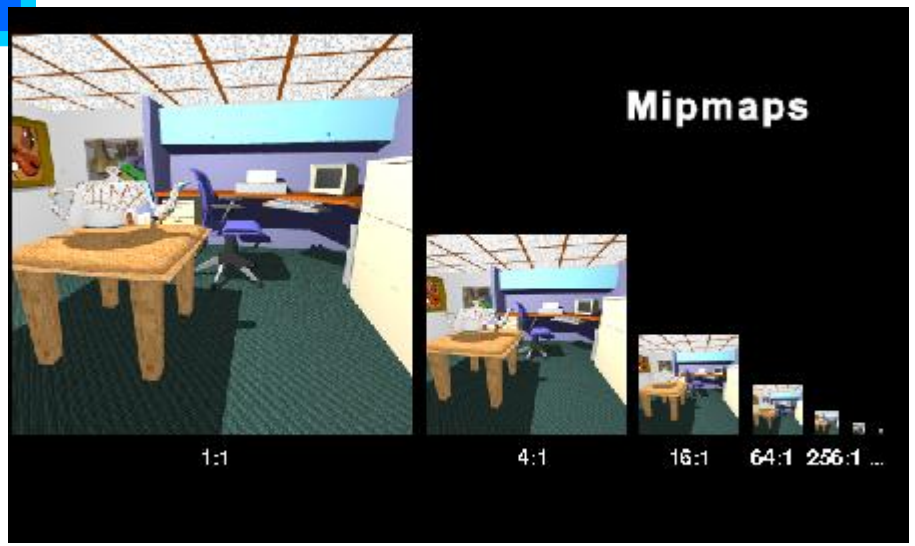
This is expensive to do if averaging is done over large areas repeatedly.





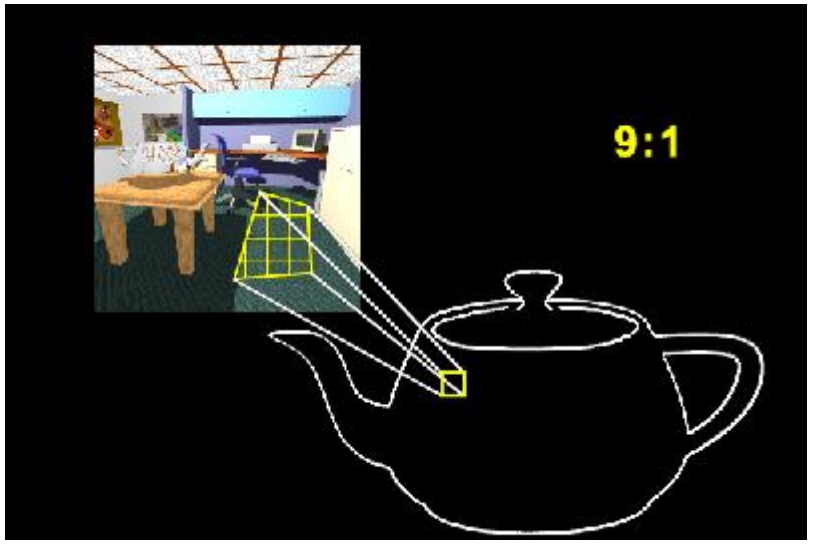
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

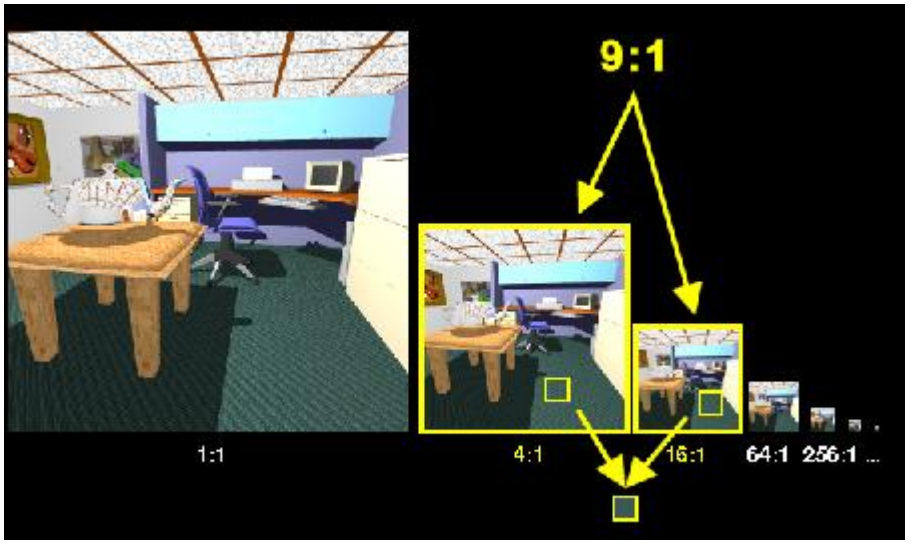
Texture Aliasing



Mipmapping alleviates this problem by pre-calculating the average colours.

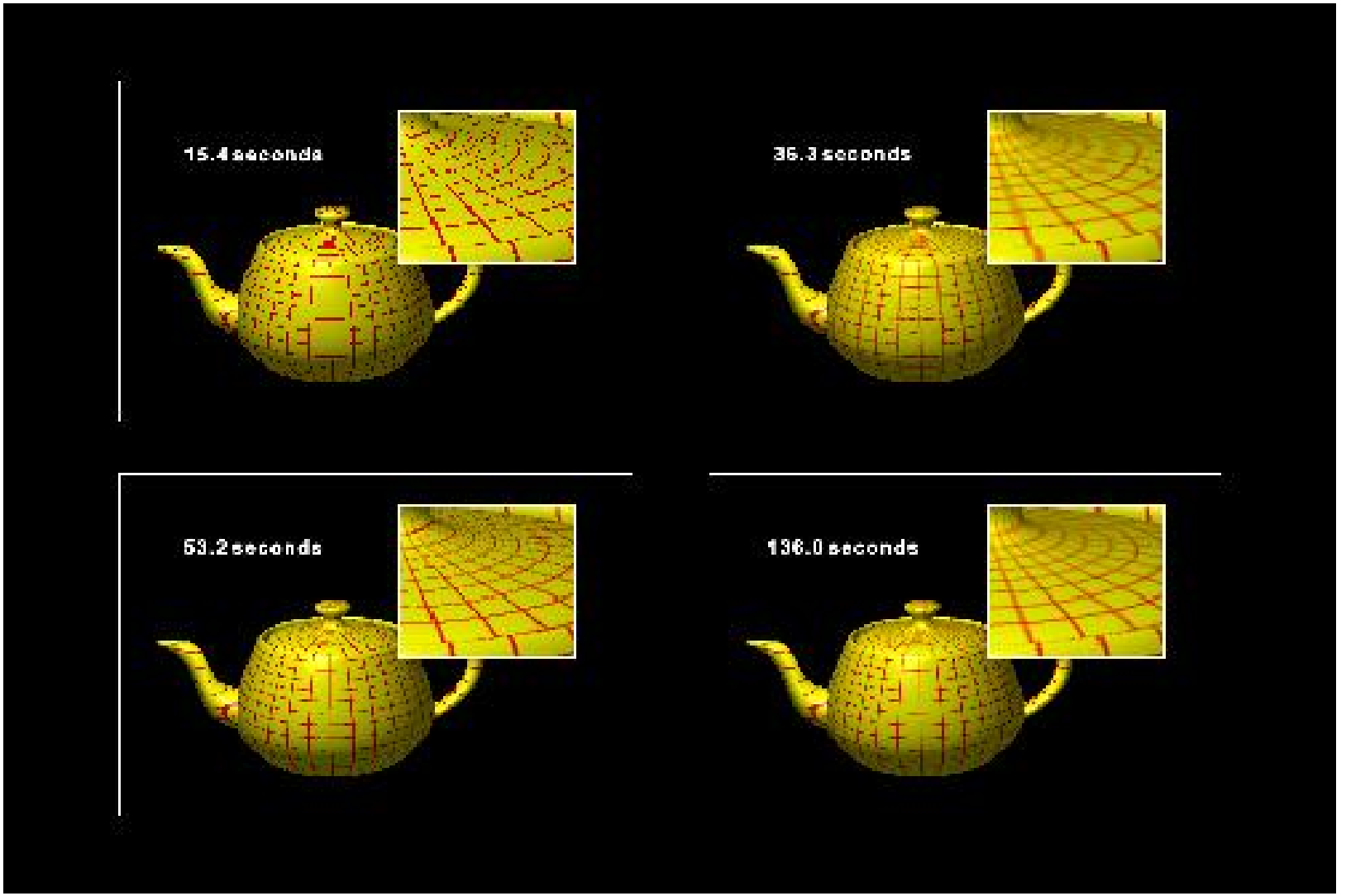
In the new, smaller texture map, each pixel contains the average of four pixels from the original texture.





http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

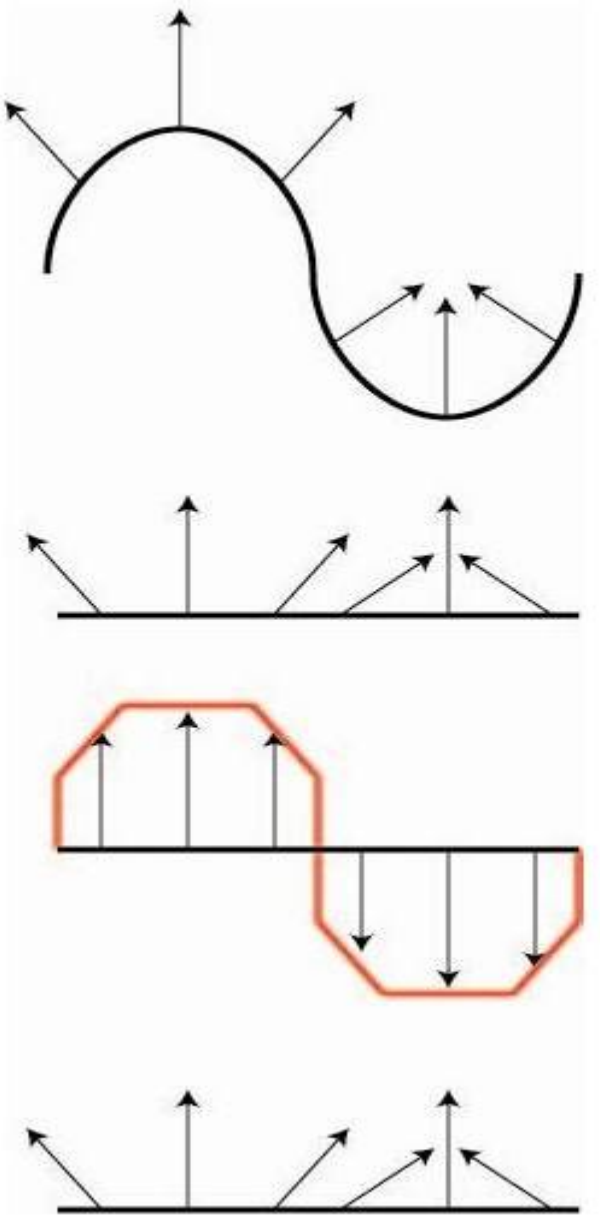
Texture Aliasing



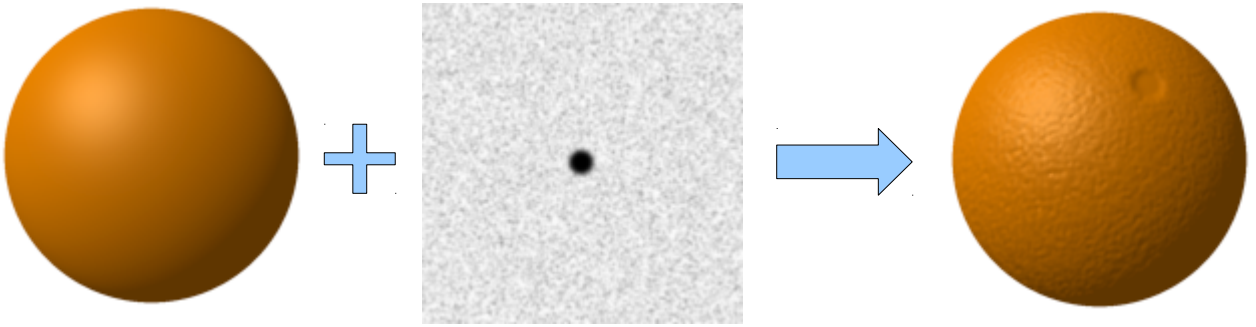
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

Surface Detail

- Ideal Geometry
- Adjust normals with a *bump map.*
- Approximate geometry with a *displacement map.*
- Replace surface normals with a *normal map.*



Bump Mapping

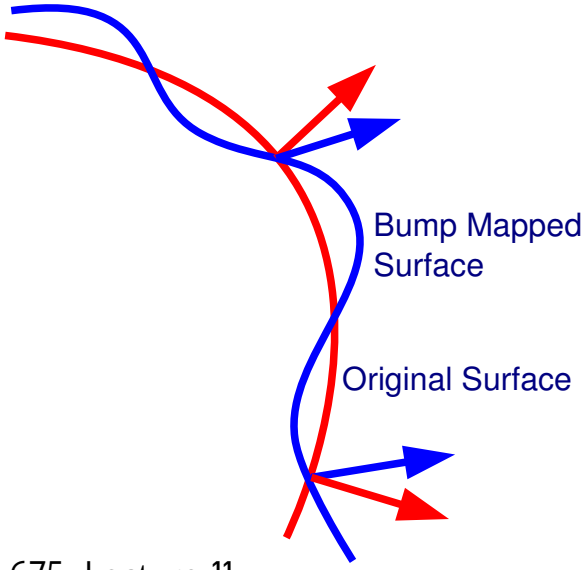


http://en.wikipedia.org/wiki/Bump_mapping

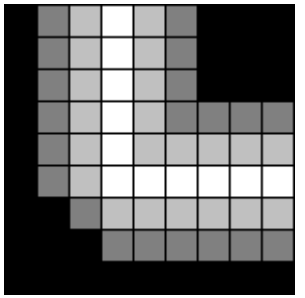
Object

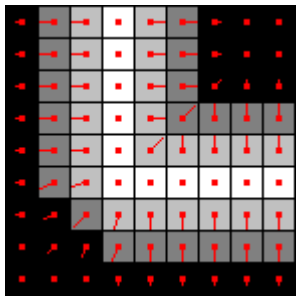
Bump Map

Bump Mapped Object



Use an array of values to perturb surface normals (calculate gradient at every image pixel and add it to the normal).





http://freespace.virgin.net/hugo.elias/graphics/x_polybm.htm

Normal Mapping

- *Bump mapping* uses a single-channel (grayscale) map to **perturb** existing normals on the surface;
  - treat grayscale map as height field
  - perturb the surface normals by the gradient of the map
  - lighting is calculated using the normal

  Normal mapping uses a multichannel (rgb) map to completely **replace** the existing normals. RGB values of each pixel corresponding to the x,y,z components of the normal vector
  - x,y,z components of the mapped normals are usually in object-space but other spaces are sometimes used
  - level of detail of output renders is limited by resolution of normal maps instead of by the number of polygons in rendered meshes!
  - runtime surface normal determination is trivial—it's just a look-up—making this technique particularly useful for real-time applications

  Limitations:

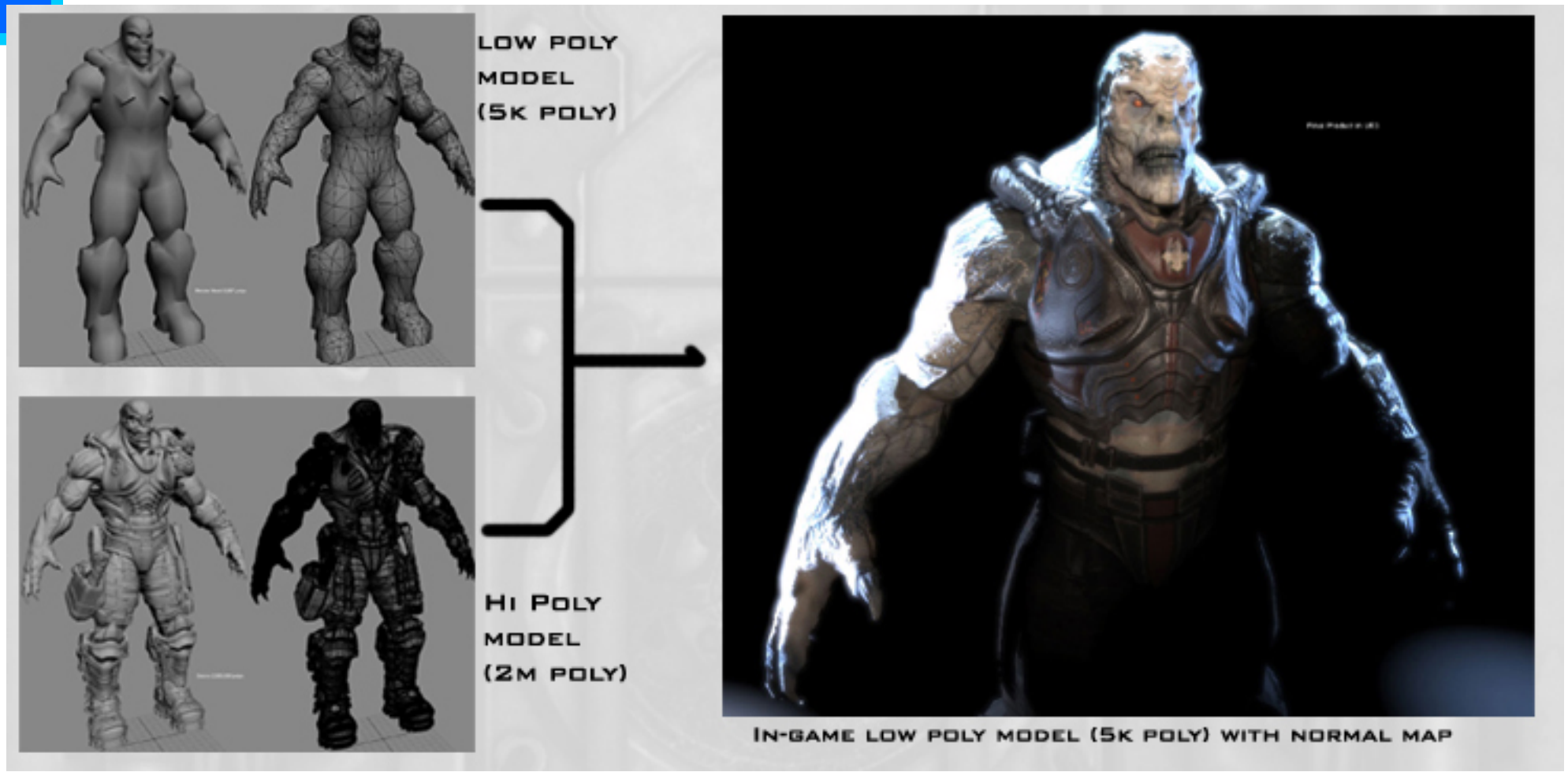
- silhouettes do not reflect added detail
- though runtime is easy, initial creation of normal maps is not straight-forward

Normal Mapping

Image courtesy of ATI/AMD

Normal map can be used to fake additional geometry.

Normal Mapping



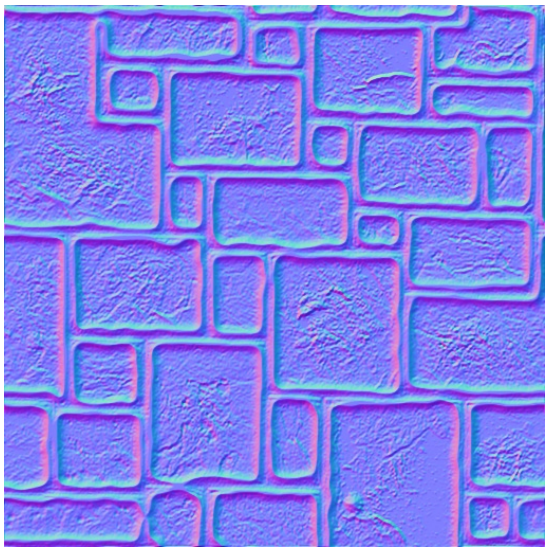
Images courtesy of www.unrealtechnology.com
Unreal Engine 3 (Gears of War, Epic Games)

Normal Mapping

- Creation of meaningful normal maps is not simple
- Unlike bump maps, normal maps can not simply be painted by hand because color in a normal map is constantly varied and r,g,b values are spatially meaningful.

Normal map for a face.

Normal map for a brick wall.



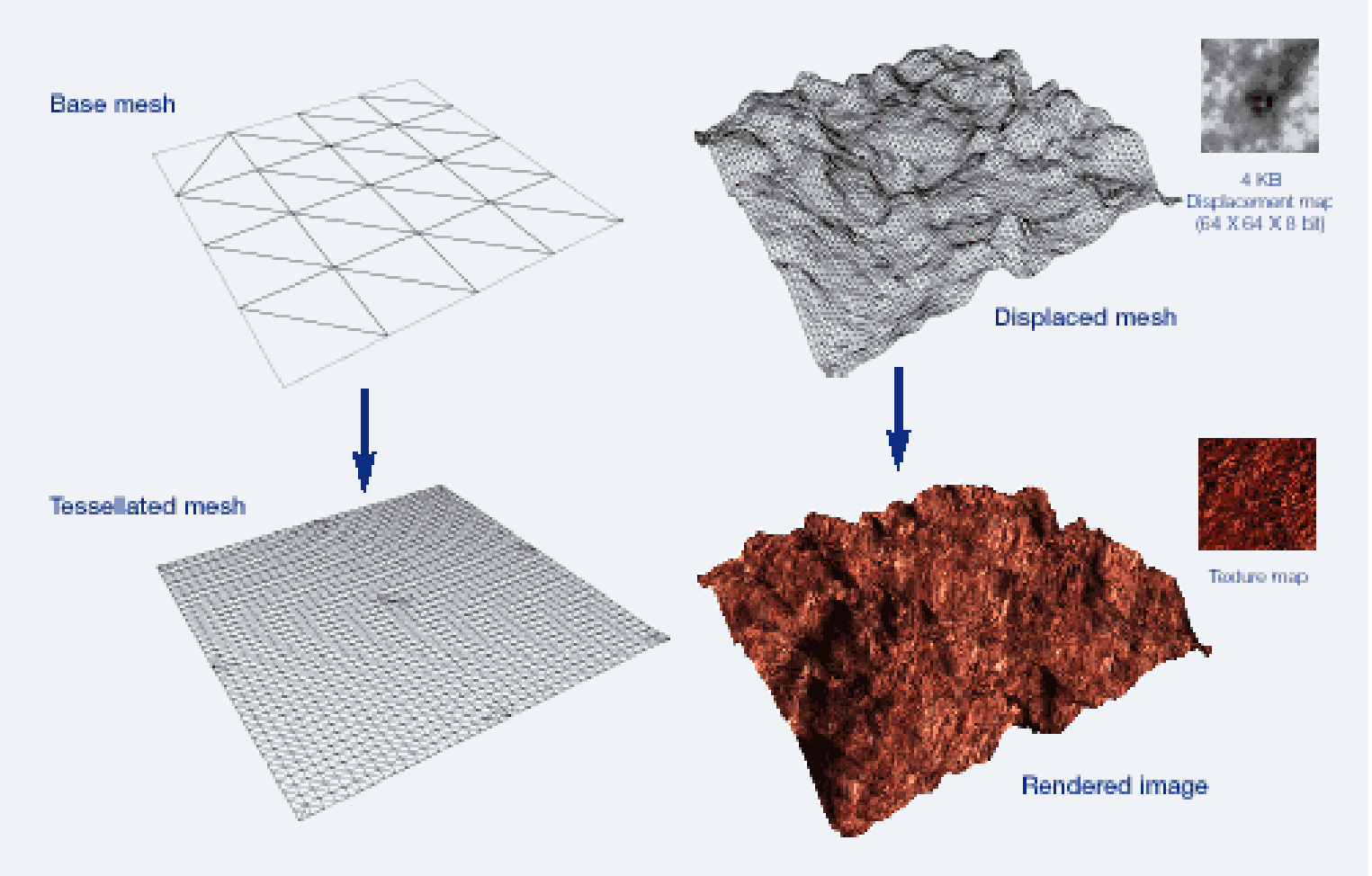
http://developer.nvidia.com/object/real-time-normal-map-dxt-compression.html

www.bitmanagement.com

- Normal maps must be generated by specialized software, such as Pixologic's Zbrush (www.pixologic.com)
- A high resolution model is created by the artist and its normals are used to generate maps for lower-res versions of the model

Displacement Mapping

- Actual geometric position of points over the surface are *displaced* along the surface normal according to the values in the texture.



http://www.xbitlabs.com/articles/video/display/matrox-parhelia.html