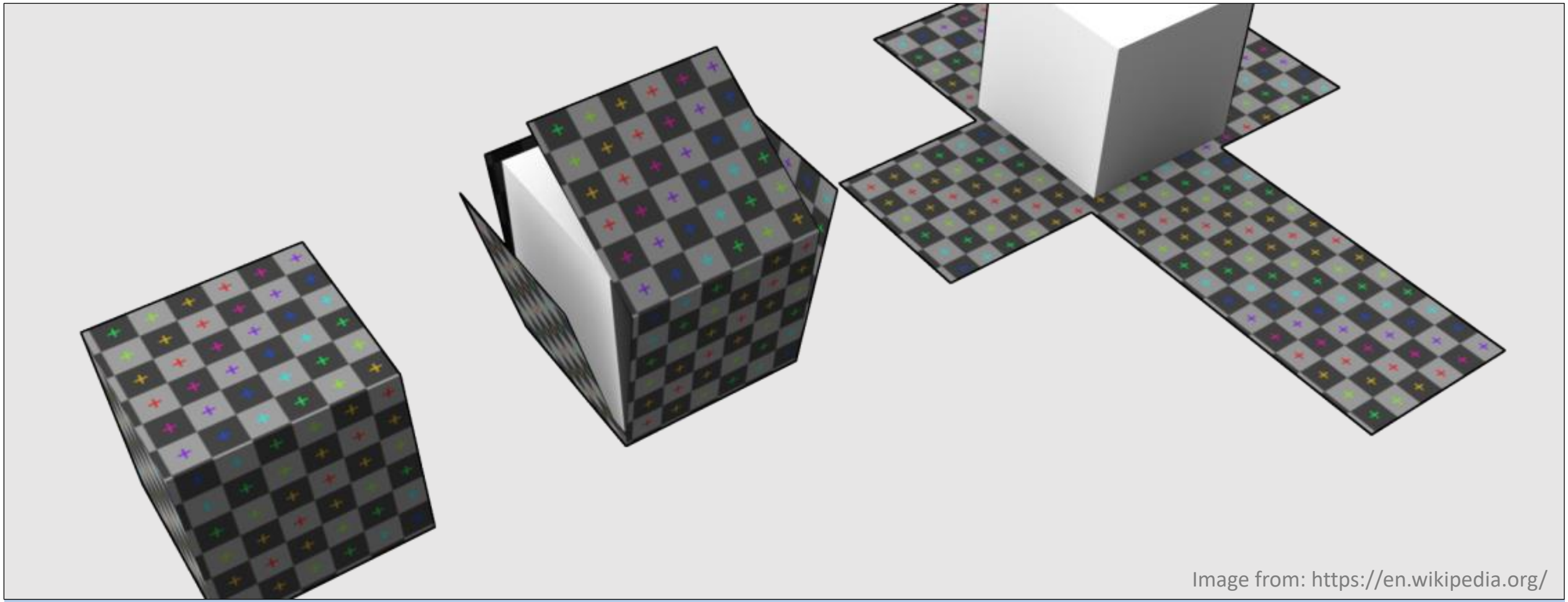


Image from: https://en.wikipedia.org/

Γραφικά Υπολογιστών

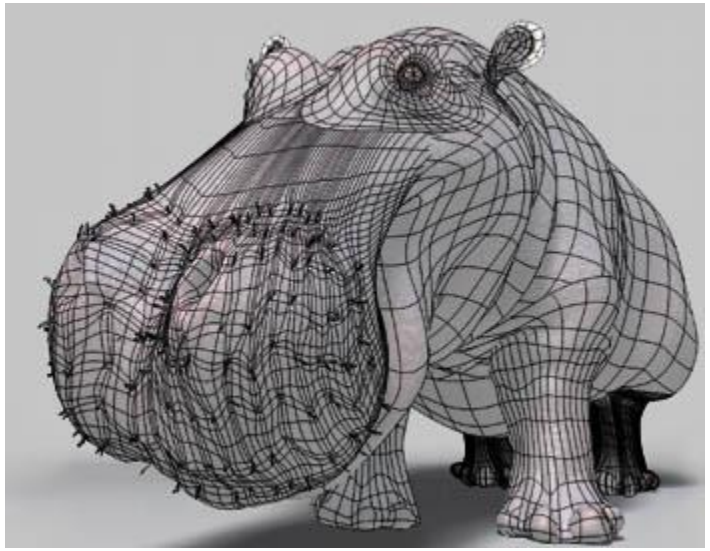
Texture mapping, Antialiasing

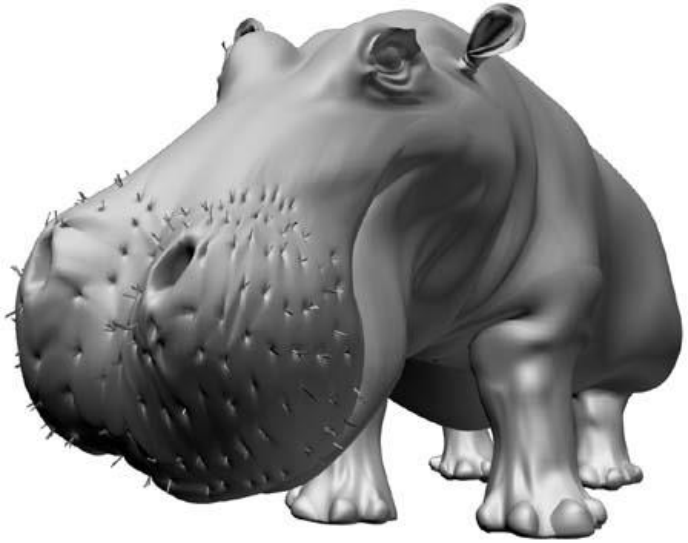
Andreas Aristidou
andarist@ucy.ac.cy
http://www.andreasaristidou.com

Overview

- Texture mapping
- Antialiasing
- Antialiasing-textures

Texture mapping





http://www.3drender.com/

Texture Mapping: *Why needed?*

- Adding details using high resolution polygon meshes is costly
- Not very suitable for real-time applications

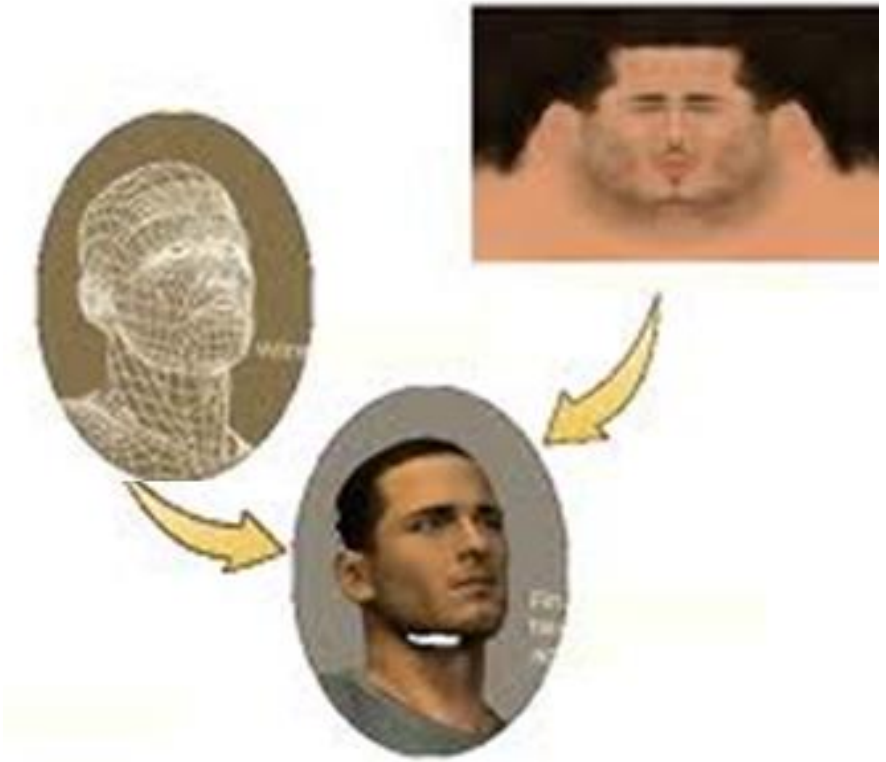
by Rami Ali Al-ashqar

Texture Mapping: *Why needed?*

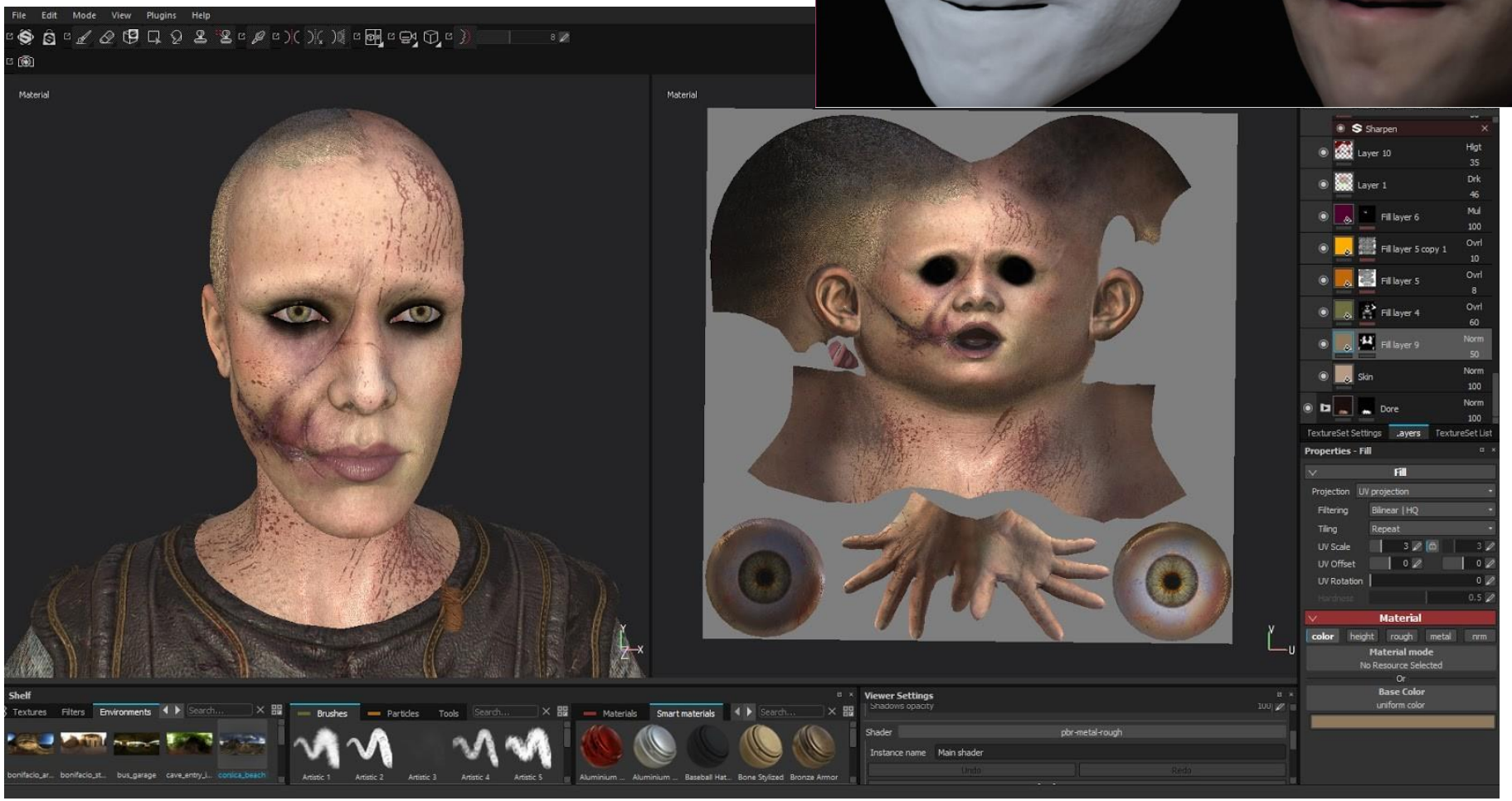
- Have seen: colour can be assigned to vertices
- But: don’t want to represent all this detail with geometry

Texture Mapping: *Why needed?*

- Method of improving surface appearance by mapping images onto the surface
- Done at the rasterization stage



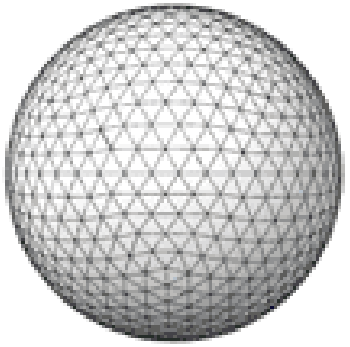
Texture Mapping: *Overv*



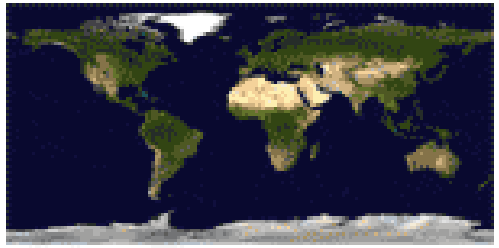
Texture Mapping: *Overview*

- Texture mapping:
  - Implemented in hardware on every GPU
  - Simplest surface detail hack, dating back to the ‘60s GE flight simulator and its terrain generator

- Technique:
  - “Paste” the texture, a photograph or pixmap (e.g., a brick pattern, a wood grain pattern, a sky with clouds) on a surface to add detail without adding more polygons
  - Map texture onto surface to assign surface color (vs. using object color) or to alter object’s surface color
  - Think of texture map as stretchable contact paper



Sphere with no texture



Texture image

Sphere with texture

Texture Mapping: *Overview*

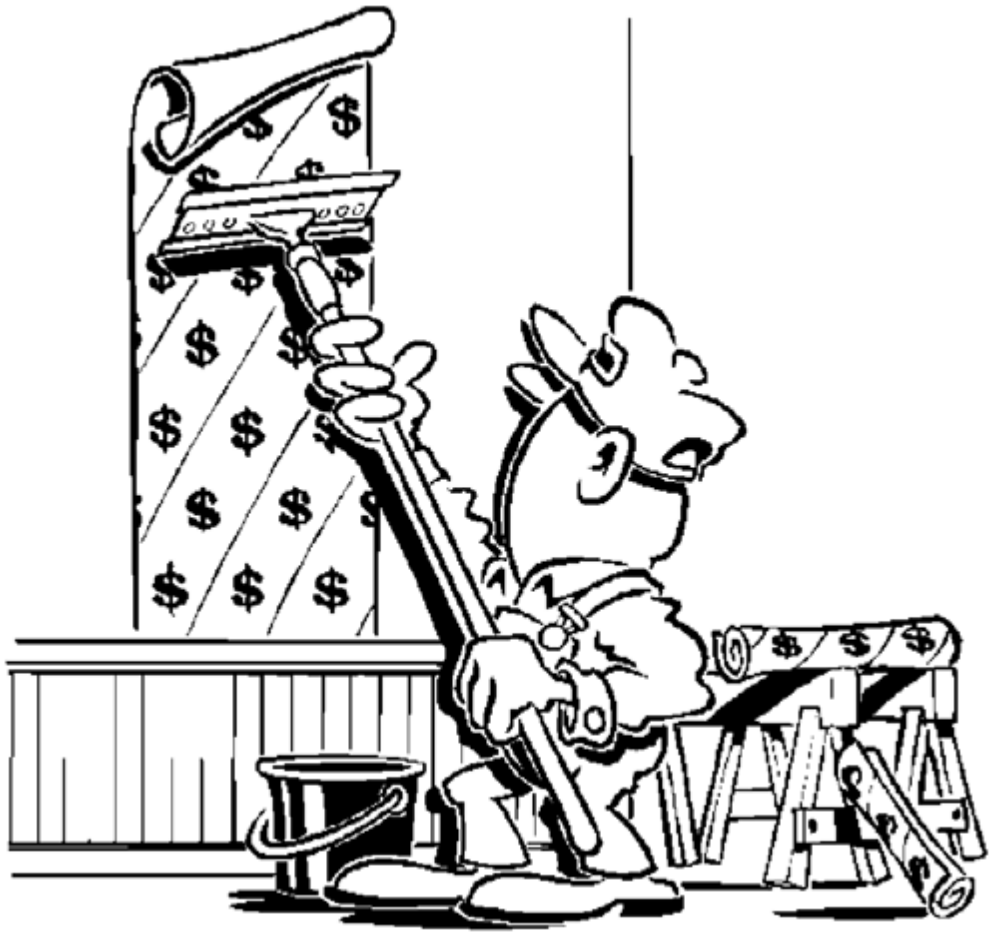
- How do we add more detail to a model?
  - Add more detailed geometry; more, smaller triangles:
    - **Pros:** Responds realistically to lighting, other surface interaction
    - **Cons**: Difficult to generate, takes longer to render, takes more memory space
  - Map a texture to a model:
    - **Pros**: Can be stored once and reused, easily compressed to reduce size, rendered very quickly, very intuitive to use, especially useful on far-away objects like terrain, sky, “billboards” (texture mapped quad) - all used extensively in videogames, etc.
    - **Cons**: Very crude approximation of real life. Surfaces still look smooth since geometry is not changed. Need to consider perspective for real effectiveness
- What can you put in a texture map? (or any other kind of map?)
  - Diffuse, ambient, specular, or any kind of color
  - Specular exponents, transparency or reflectivity coefficients
  - Surface normal data (for normal mapping or bump mapping – stay tuned)
  - Projected reflections or shadows

Texture Mapping: *Overview*

- A function is a mapping
  - Takes any value in the domain as an input and outputs ("maps it to") one unique value in the co-domain.
- Mappings up to now:
  - Linear transformations with matrices
    - Local to world
    - World to camera
    - Etc…
- Mapping a texture:
  - Take points on the surface of an object (domain)
  - Return a corresponding entry in the texture (co-domain)

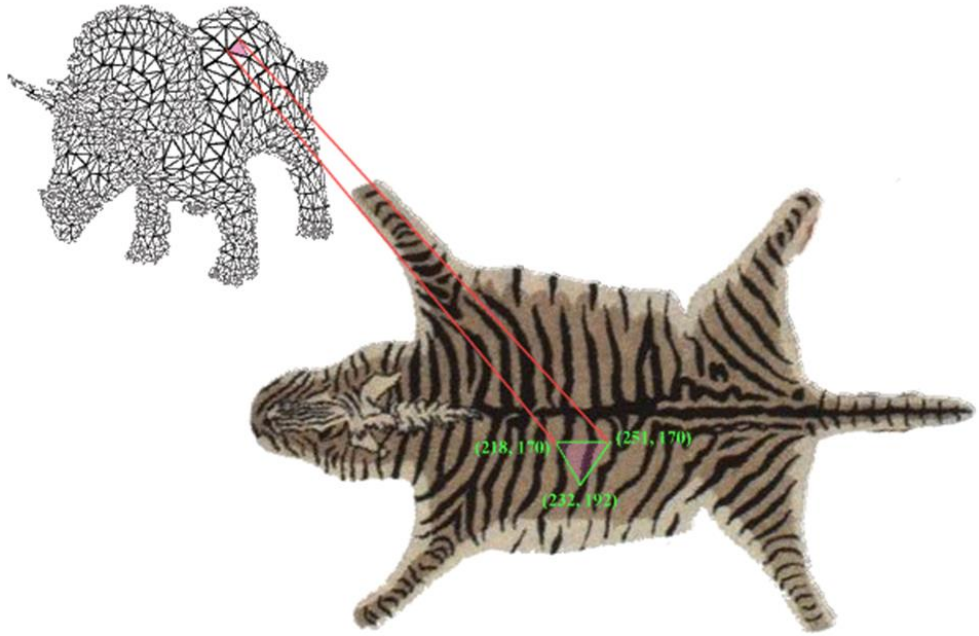
Principles of Texture Mapping

- Increase the apparent complexity of simple geometry
- Efficient packing of flat detail
- Like wallpapering or gift-wrapping with stretchy paper



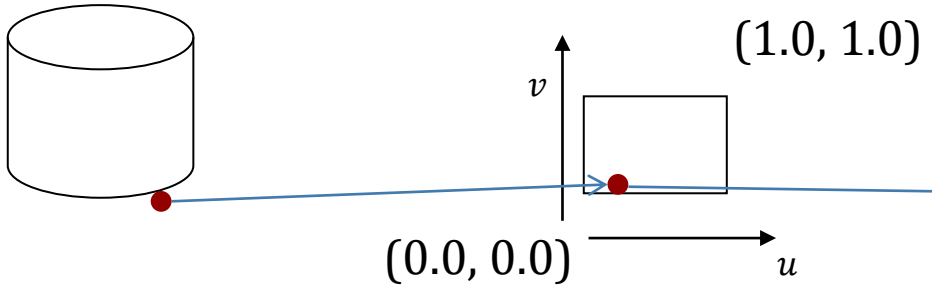
Principles of Texture Mapping

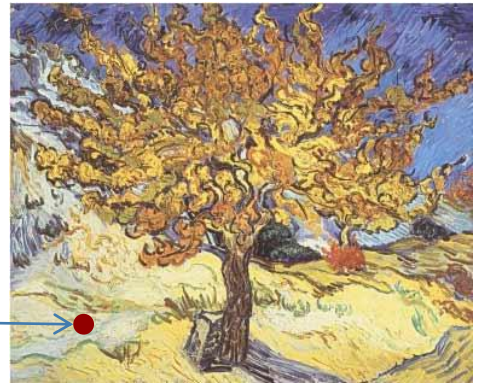
- For each triangle in the model, **establish a corresponding region in the texture image**
- Image has much higher resolution than mesh
- During the rasterization, color the surface by that of the texture
- **UV coordinates:** the 2D coordinates of the texture map



Texture Mapping Technique

- Texture mapping is process of mapping a geometric point in space to a value (color, normal, other…) in a texture map of arbitrary width and height
  - Our goal is to map any arbitrary object geometry to a texture map
  - Done in two steps:
    - Map a point on object to a point on unit square (a proxy for the actual texture map)
    - Map unit square point to point on texture



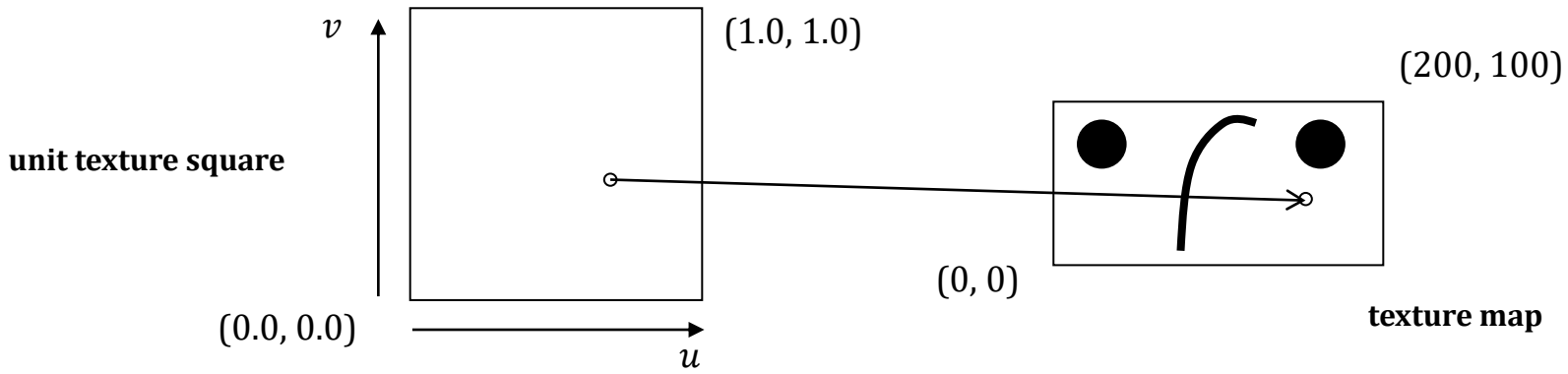


Van Gogh

  - Second mapping much easier, we'll cover it first – both maps based on proportionality
  - This 2D ***uv*** coordinate system is unrelated to the camera's 3D ***uvw*** coordinate system!
  - Here, the ***uv*** unit square is oriented with (0,0) in the bottom corner. It could have (0,0) in upper left; the choice is arbitrary.

Texture Mapping Technique

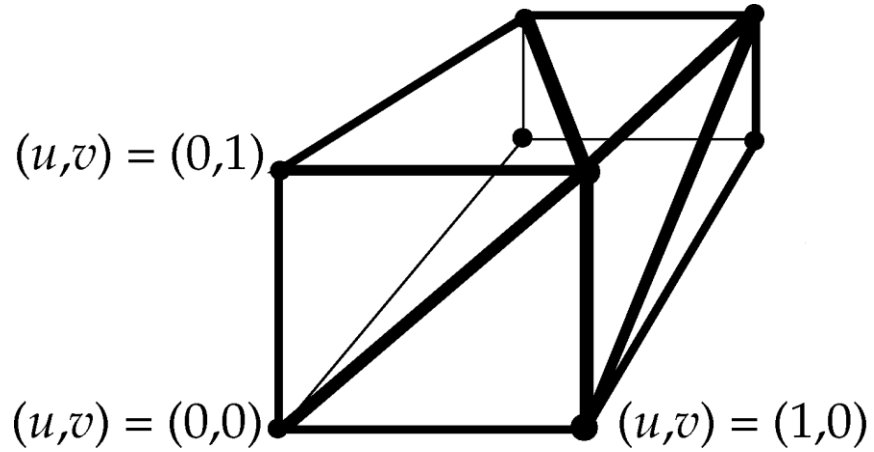
- Mapping a point ***(u, v)*** in unit square to a texture of arbitrary width ***w*** and height ***h*** :
  - Corresponding point on texture map is proportional on each axis

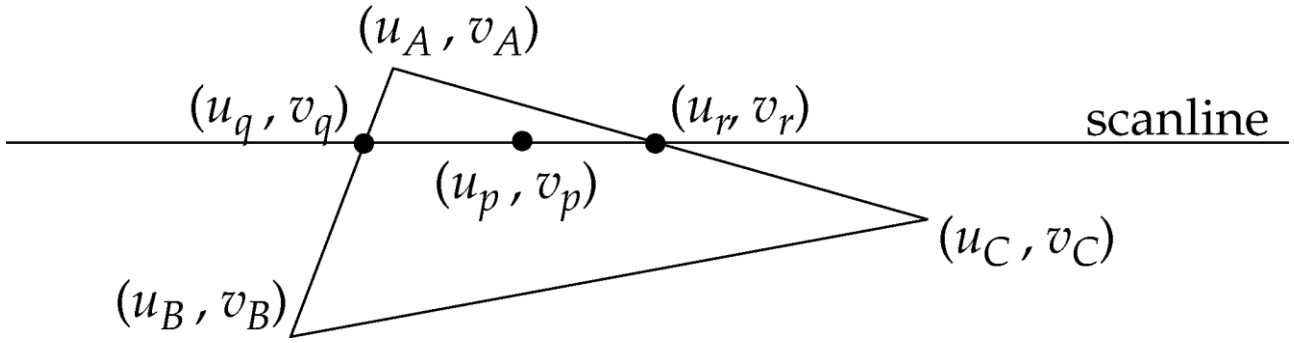


  - Above: $(0.0, 0.0) \rightarrow (0, 0)$; $(1.0, 1.0) \rightarrow (200, 100)$; $(.7, .45) \rightarrow (140, 45)$
  - Once you have coordinates for texture, just look up color of texture at these coordinates
  - Coordinates not always a discrete (int) point on texture as they are mapped from points in continuous ***uv*** space. May need to average neighboring texture pixels (i.e., filter)

Texture Mapping Technique

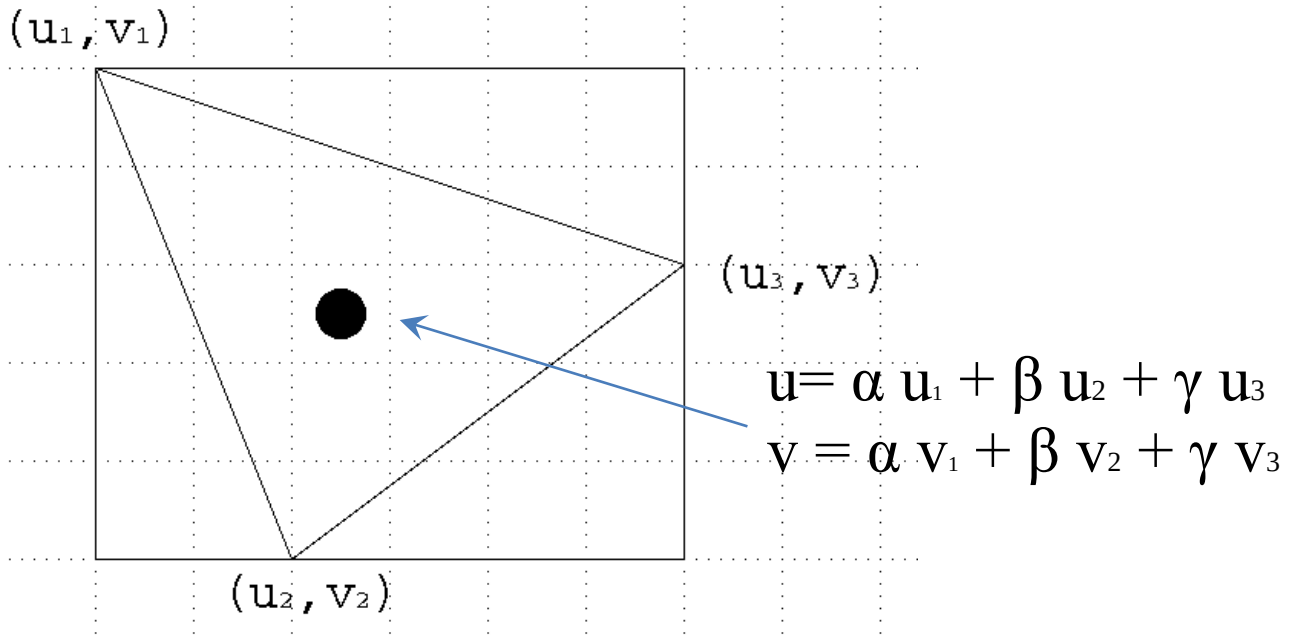
- Texture mapping individual polygons
  - (u, v) texture coordinates are pre-calculated and specified per vertex
  - Vertices may have different texture coordinates for different faces
  - Texture coordinates are linearly interpolated across polygon, as usual





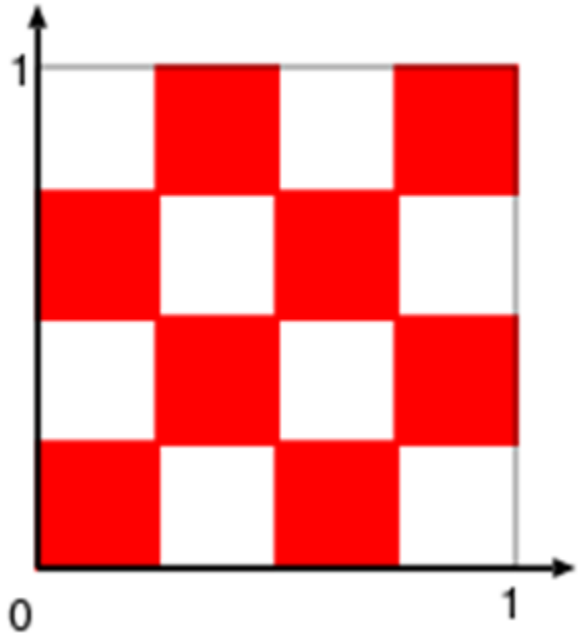
Texture Mapping: *Example*

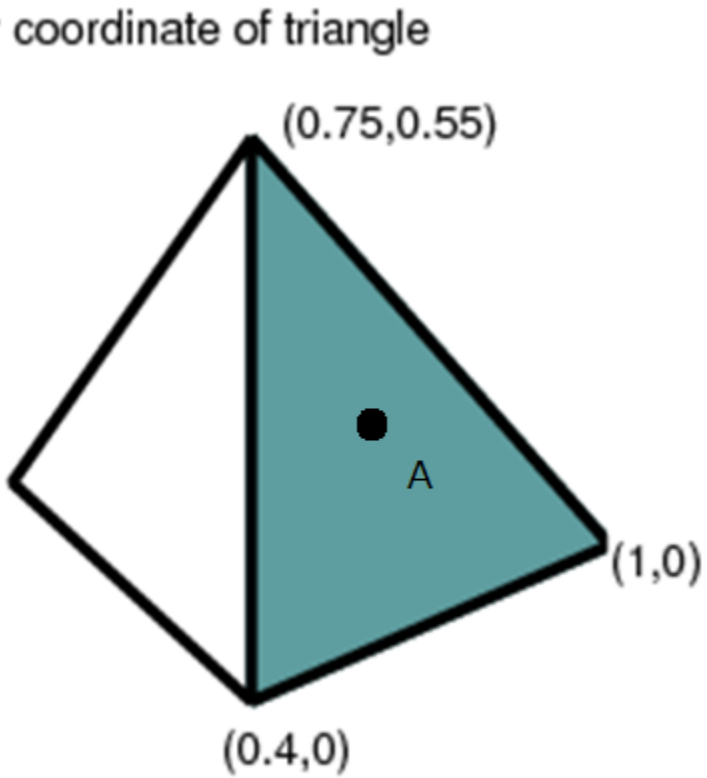
- To compute the UV inside the triangle, interpolate those at the vertices



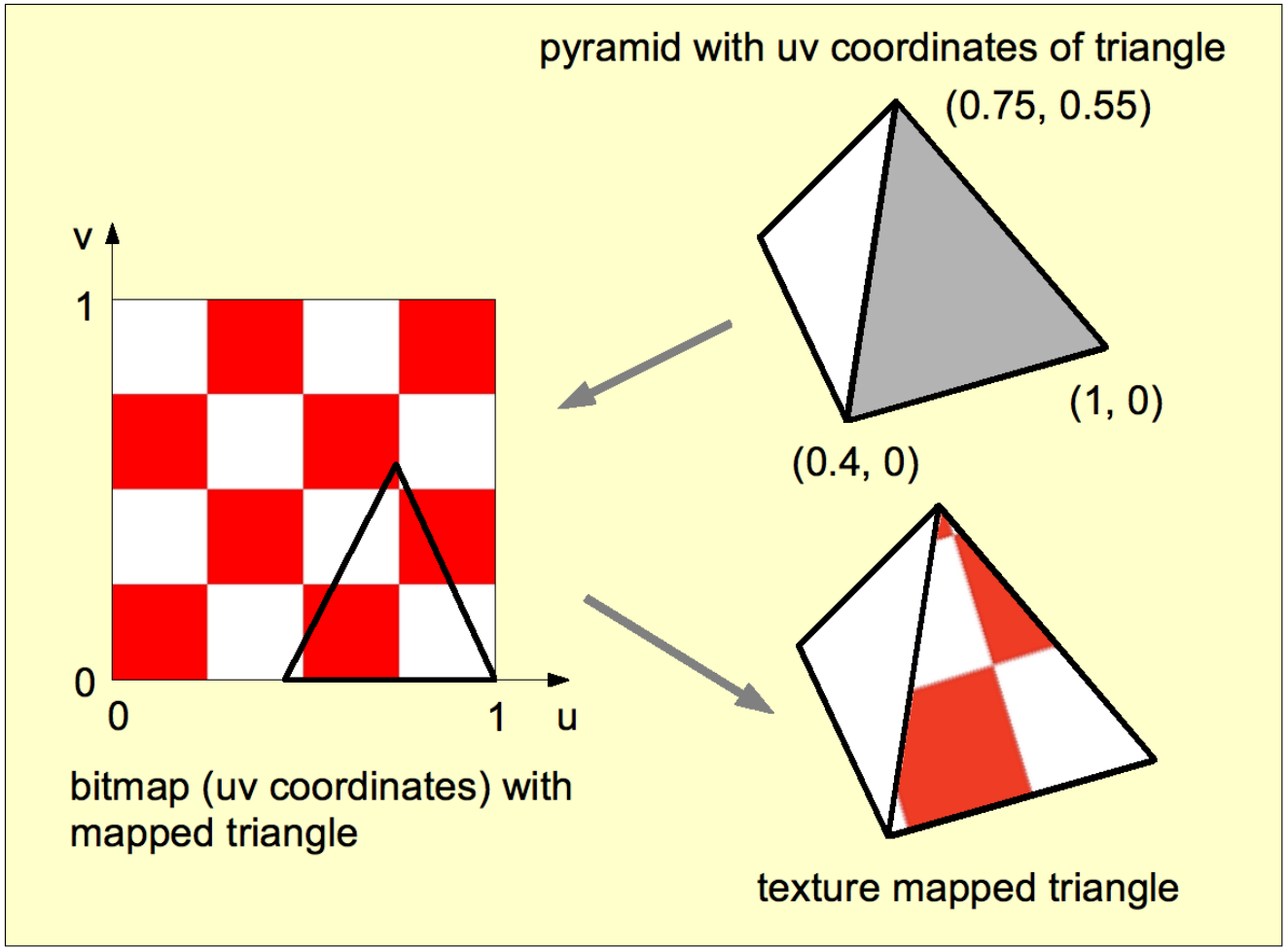
Texture Mapping: *Example*

- What is the color of the 3 vertices of the triangle?
- How will the textured triangle look like?





Texture Mapping: *Example*



How to Produce a UV Mapping?

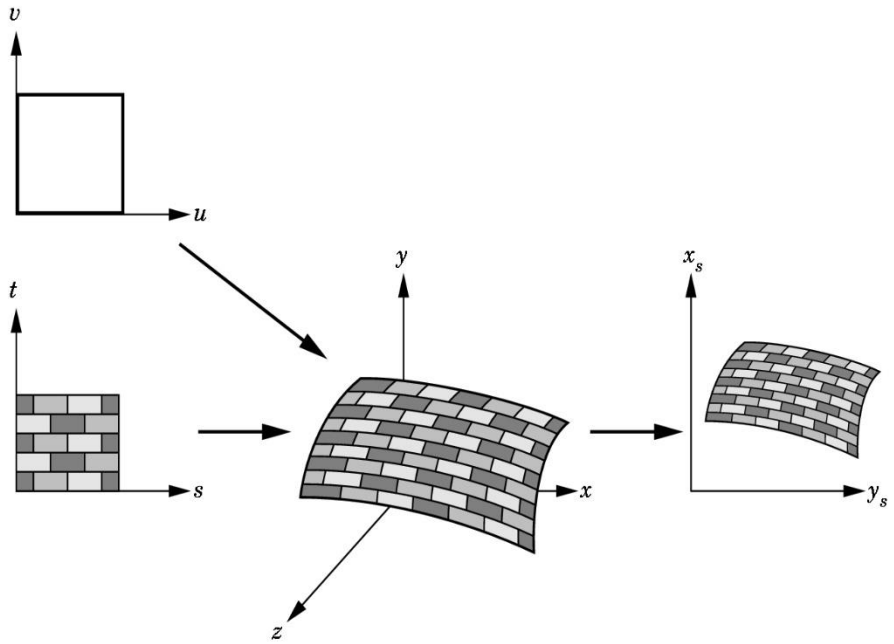
- Use common mappings
  - Orthogonal, cylindrical, spherical
- Capturing the real data
  - Obtain the color as well as the 3D shape
- Manually specify the correspondence
  - Using graphical tools
  - Using automatic segmentation, unfolding

Common Texture Coordinate Mappings

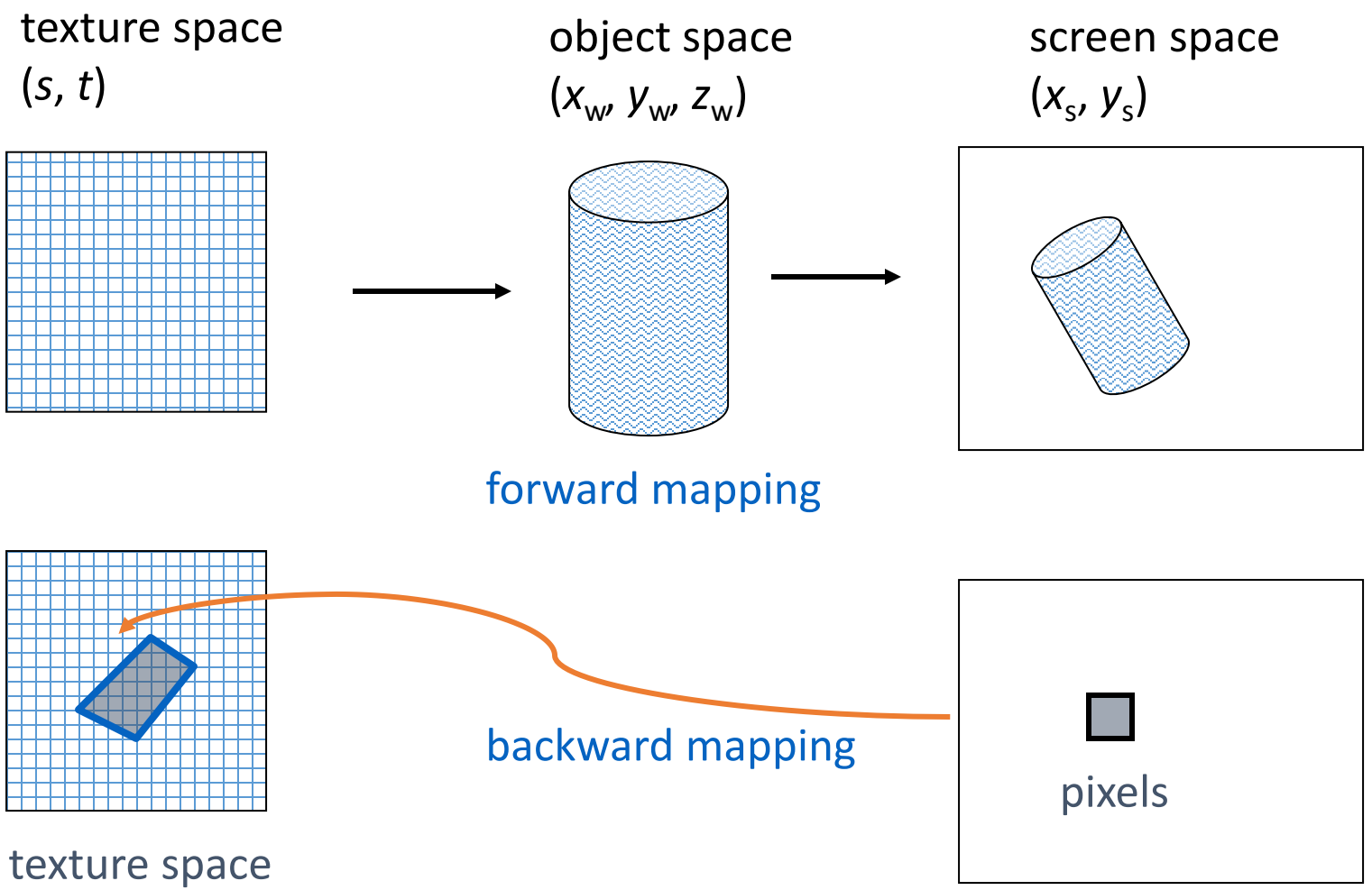
- Orthogonal
- Cylindrical
- Spherical

Introduction: *Two-Dimensional Texture Mapping*

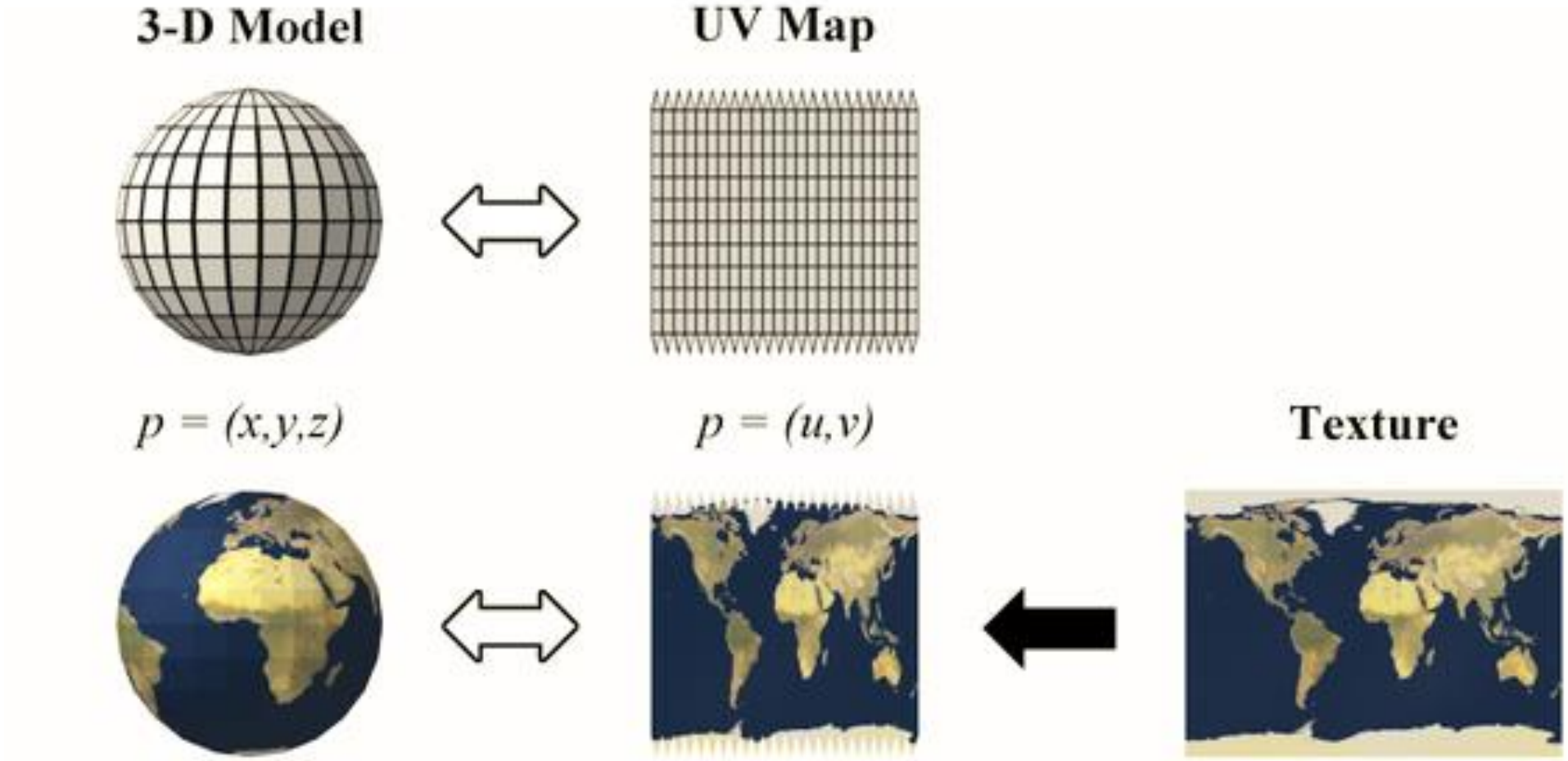
- Texture space (s, t) : Image [0,1] × [0,1]
  - Texel : Texture element. pixel in the texture
- Object space (x_w, y_w, z_w) is 3 dimensional
- Screen space (x_s, y_s) is 2D



Introduction: *Texture Mapping Methods*



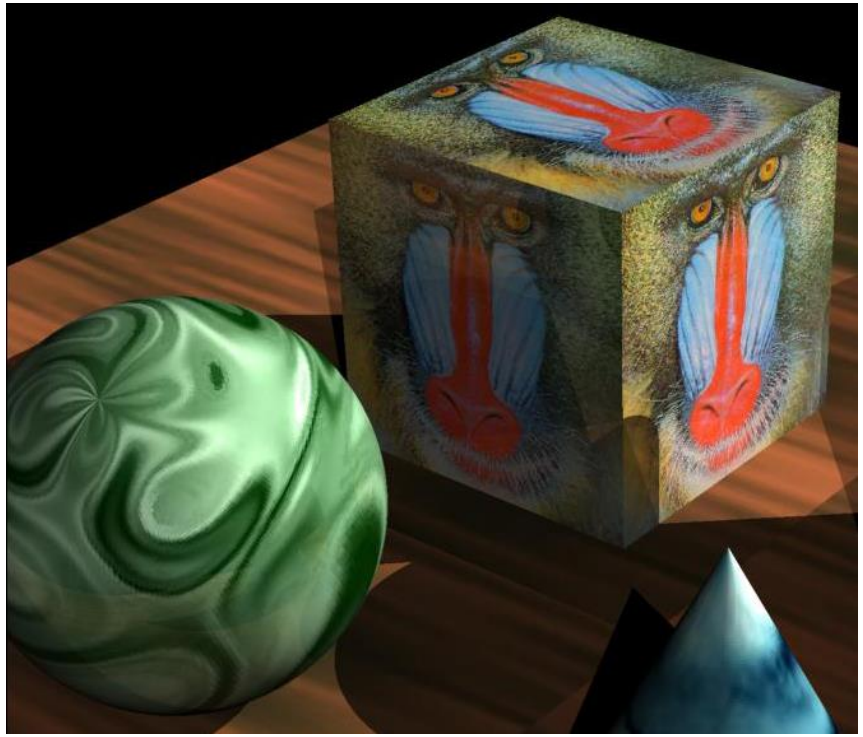
Common Texture Coordinate Mappings



Texture Mapping Technique

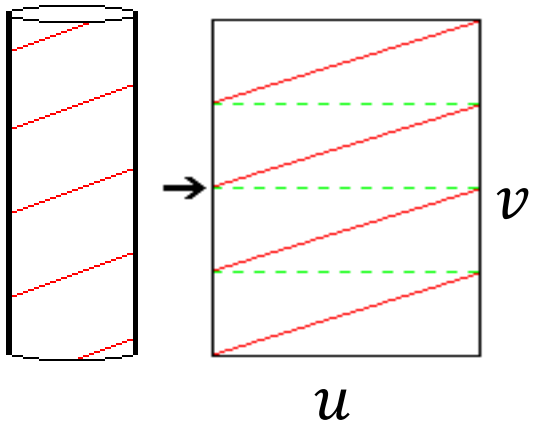
Mapping from point on object to (u, v) square

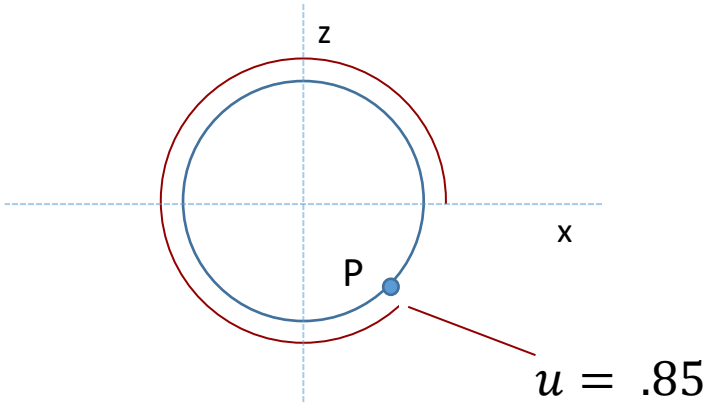
- Texture mapping solids
  - Using ray tracing, get an intersection point (x, y, z) in object space
  - Need to map this point to a point on the (u, v) unit square, so we can map that to a texture value
  - Three easy cases: planes, cylinders, and spheres
  - Easiest to compute the mapping from (x, y, z) coordinates in **object space** to (u, v)
  - Can cause unwanted texture scaling (use filters!)
  - Texture filtering is an option in most graphics libraries
  - OpenGL allows you to choose filtering method
    - GL_NEAREST: Picks the nearest pixel in the texture
    - GL_LINEAR: Weighted average of the 4 nearest pixels

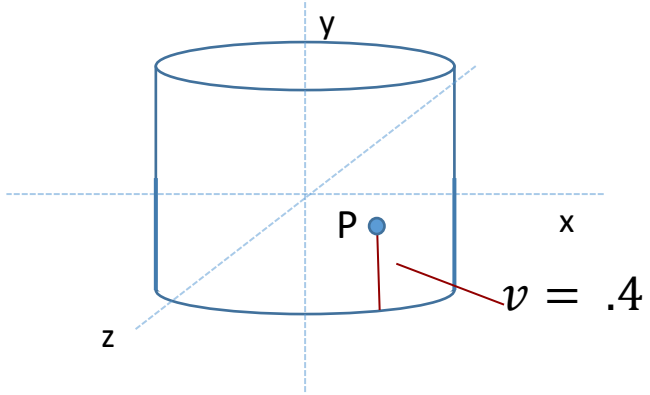


Texture Mapping Technique

- How to texture map **cylinders** and **cones**:
  - Given a point **P** on the surface:
    - If it's on one of the caps, map as though the cap is a plane
    - If it's on the curved surface:
      - Use position of point around perimeter to determine *u*
      - Use height of point to determine *v*





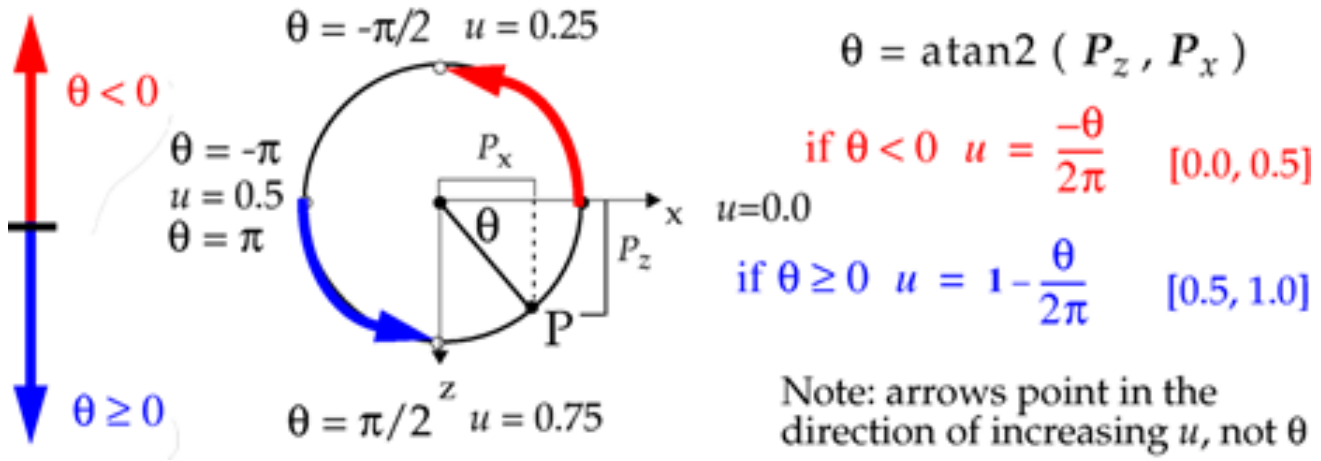


- Mapping ***v*** is trivial: [-.5, .5] for unit cylinder gets mapped to [0.0, 1.0] just by adding .5

http://www.syvum.com/iosundry/teasers/cylind1b.gif

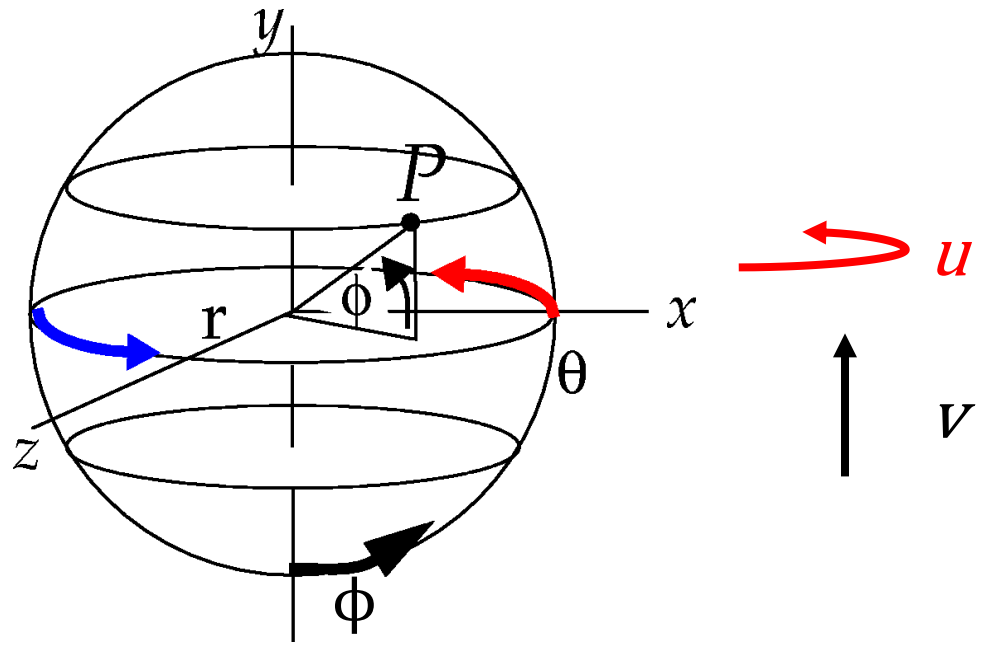
Texture Mapping Technique

- Computing u coordinate for **cones** and **cylinders**:
  - Must map all points on perimeter to [0, 1], going CCW in normal polar coordinate system (see arrows)
  - Note where positive first quadrant is, based on *z* pointing down in top view of XYZ space!
  - Easiest way is to say $u = \frac{\theta}{2\pi}$, but computing θ can be tricky
  - $\text{atan}(\frac{z}{x})$ yields $\theta \in (\frac{-\pi}{2}, \frac{\pi}{2})$, mapping two perimeter positions to the same θ value
    - Example: $\text{atan}(\frac{1}{1}) = \text{atan}(\frac{-1}{-1}) = \frac{\pi}{4}$
  - $\text{atan2}(z, x)$ yields $\theta \in (-\pi, \pi)$
    - But isn't continuous -- see diagram
    - The 2 in atan2 just means 2nd form



Texture Mapping Technique

- Texture mapping for **spheres**:
  - Find (u, v) coordinates for **P**
  - We compute u the same we do for cylinders and cones: distance around perimeter of circle
  - At poles, $v = 0$ or $v = 1$, there is a singularity. Set u to some predefined value. (.5 is good)
  - v is a function of the latitude ϕ of P



$$\phi = \sin^{-1}\frac{P_y}{r}$$

r = radius

$$v = \frac{\phi}{\pi} + \frac{1}{2}$$

$$-\frac{\pi}{2} \le \phi \le \frac{\pi}{2}$$

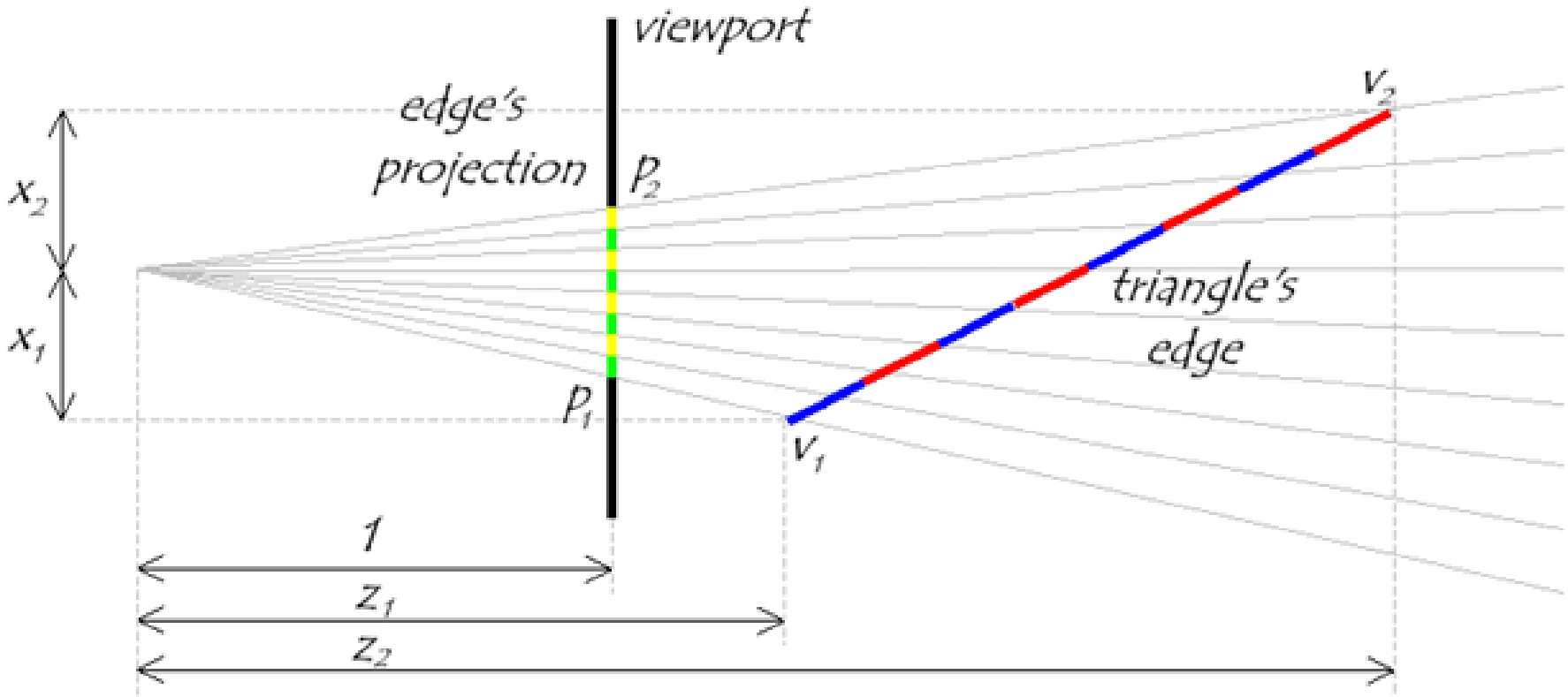
Problems with Linear Interpolation of UV Coordinates

- Linear interpolation in screen space:

texture source | what we get | what we want

Why Does it Happen?

- Uniform steps on the image plane does not correspond to uniform steps in the original 3D scene

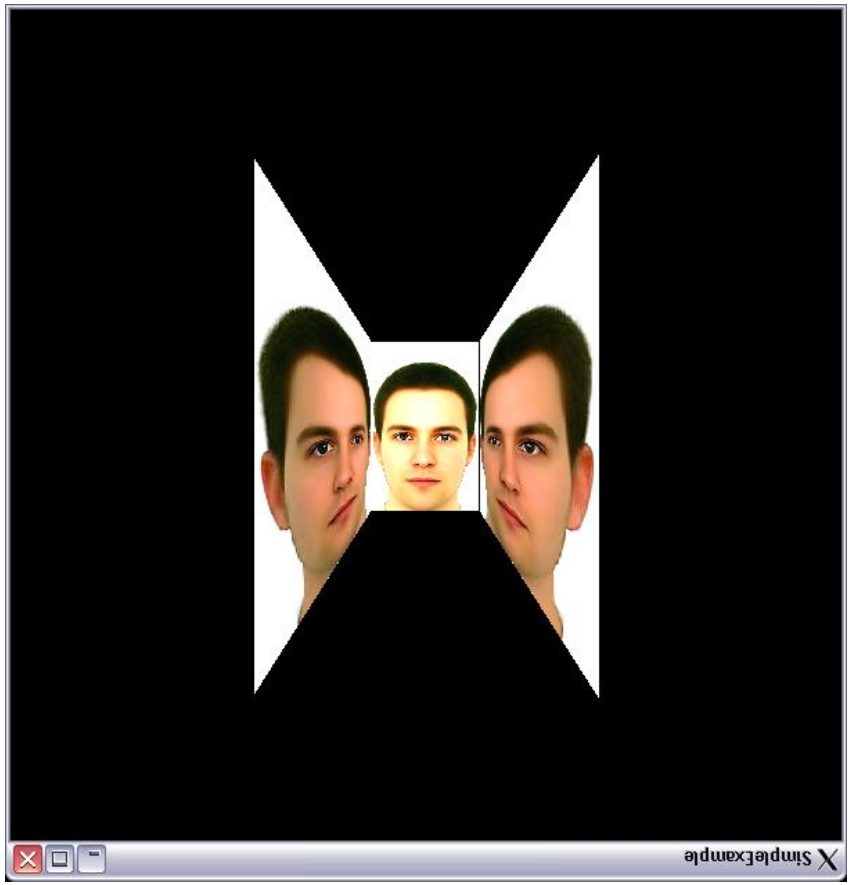


Solution : Hyperbolic Interpolation

- (u,v) cannot be linearly interpolated, but 1/w and (u/w, v/w) can
- Compute (u,v) by dividing the interpolated (u/w, v/w) by the interpolated 1/w
- w is the last component after the canonical view transformation
- A w is computed for every 3D vertex

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

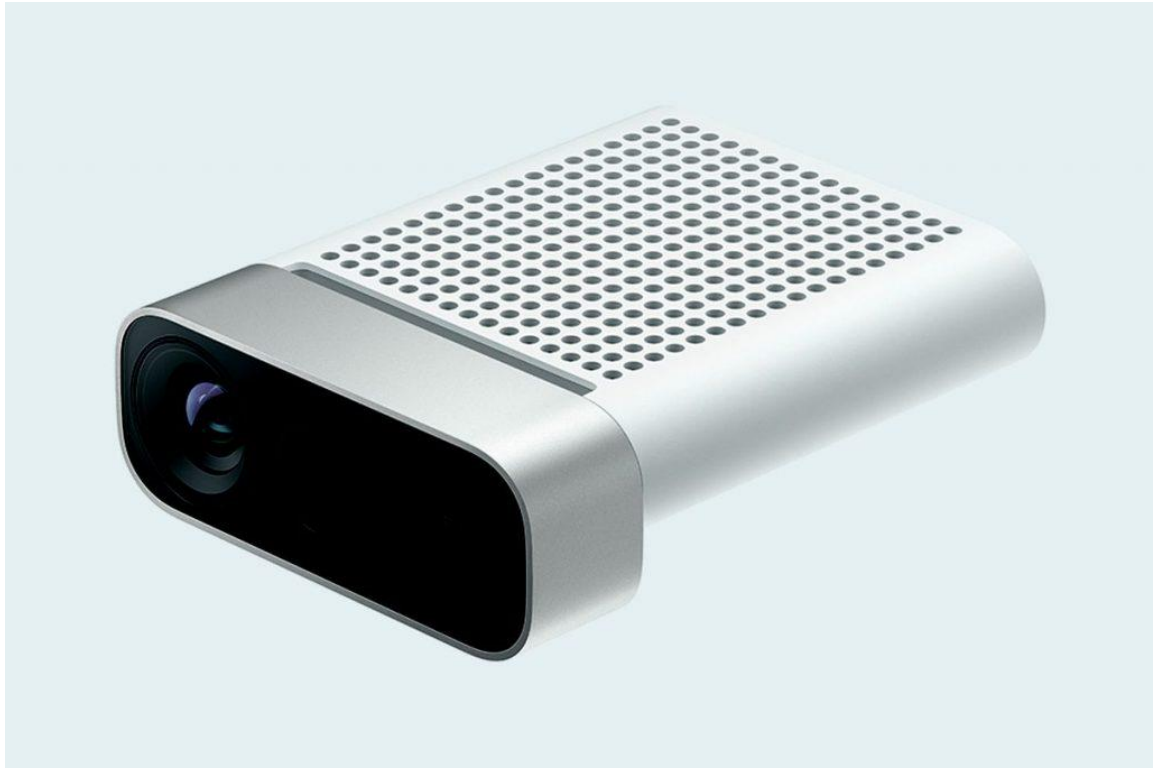
Texture Mapping Examples



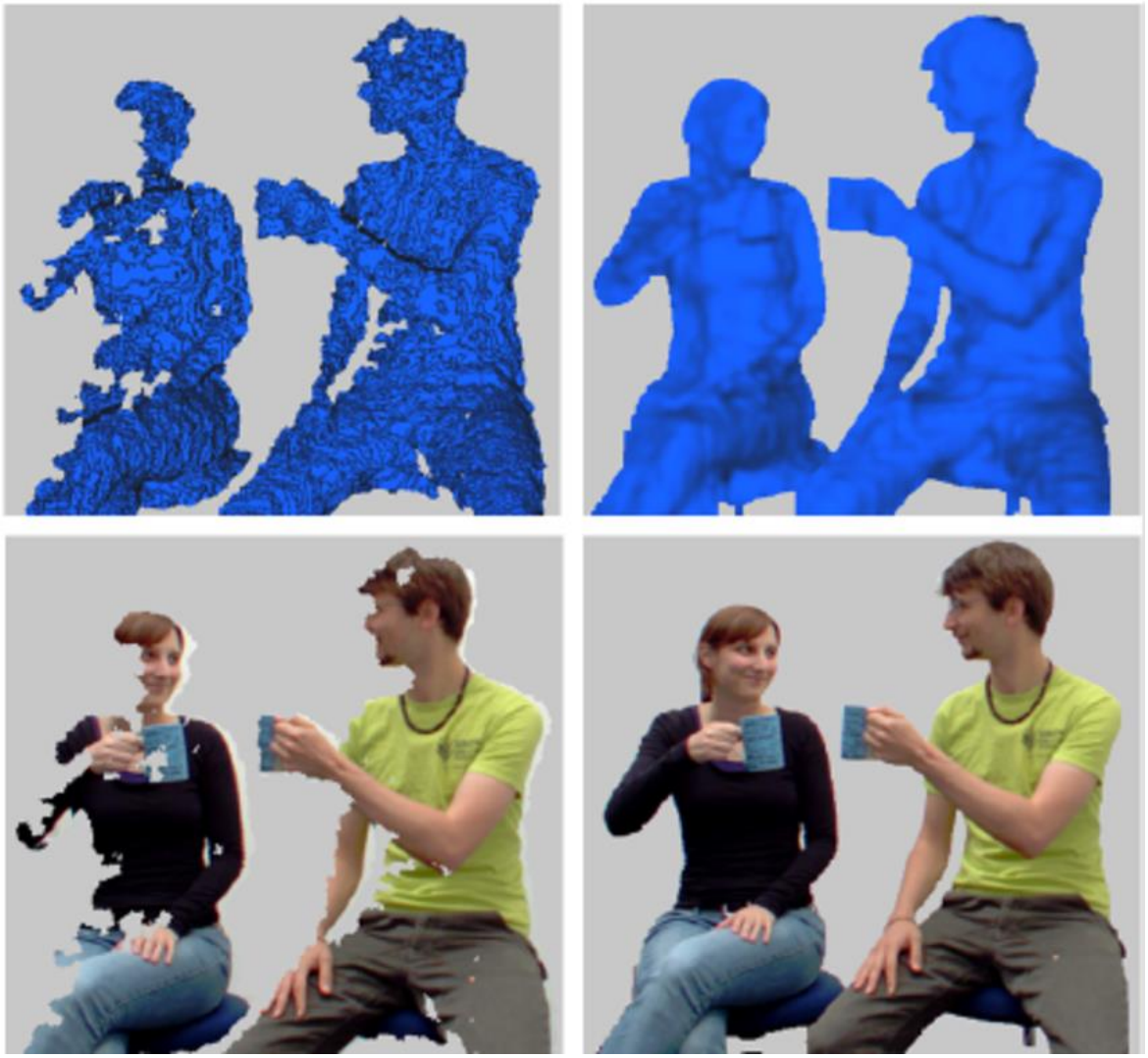
- Linear interpolation vs. Hyperbolic interpolation
- Two triangles per square

UV Mapping: *Capture Real Data*

- Capture the depth as well as the color

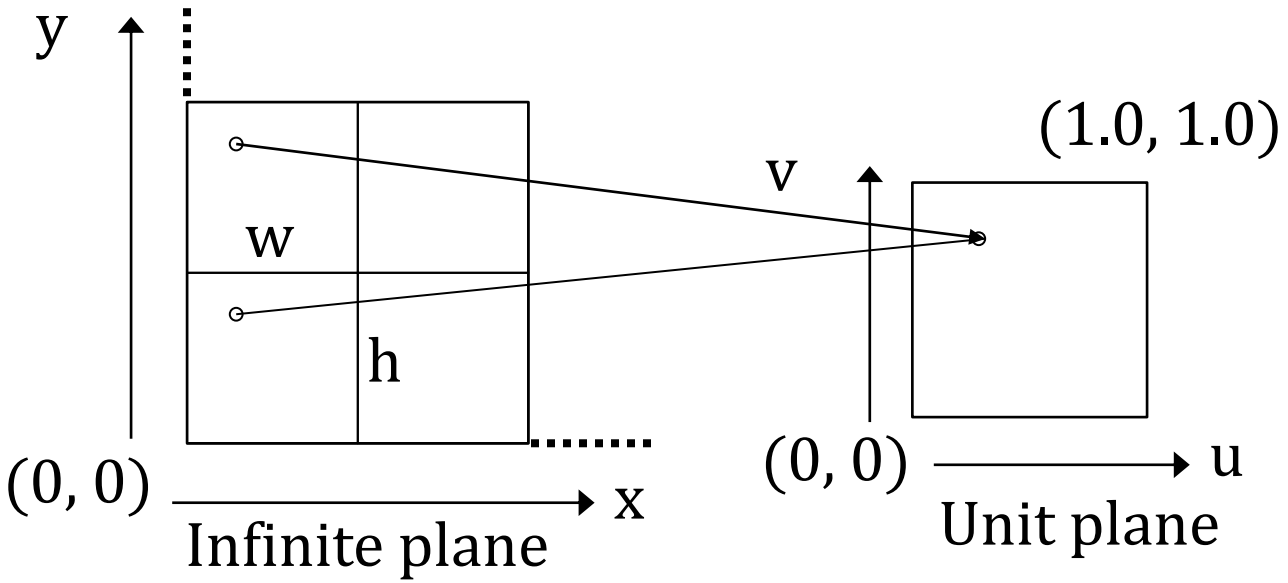


Microsoft Kinect Azure



Texture Mapping Technique

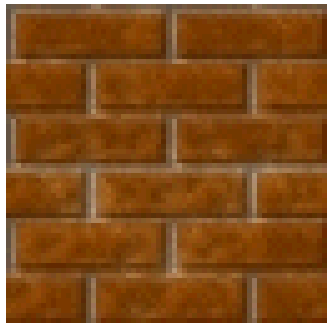
- Texture mapping large quads:
  - How to map a point on a very large quad to a point on the unit square?
  - **Tiling**: texture is repeated over and over across infinite plane
  - Given coordinates (x, y) of a point on an arbitrarily large quad to tile with quads of size (w, h), the (u, v) coordinates on the unit square are:



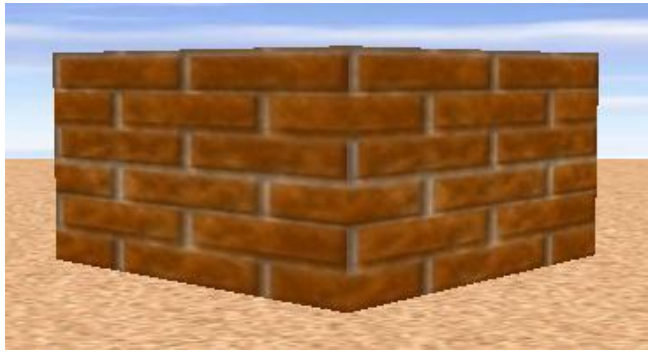
$$(u, v) = \left(\frac{(x \,\%\, w)}{w}, \frac{(y \,\%\, h)}{h} \right)$$

Texture Mapping Style - Tiling

- We want to create a brick wall with a brick pattern texture
  - A brick pattern is very repetitive, we can use a small texture and “tile” it across the wall

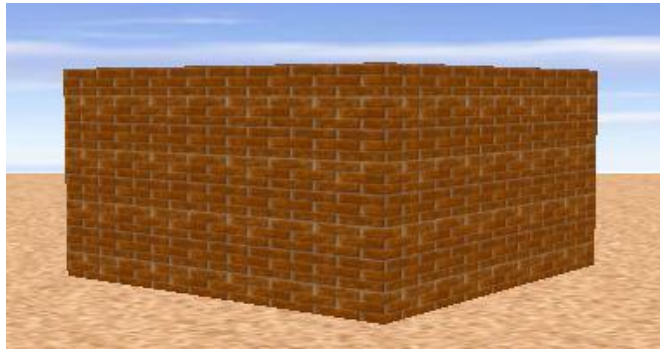


Texture



Without Tiling

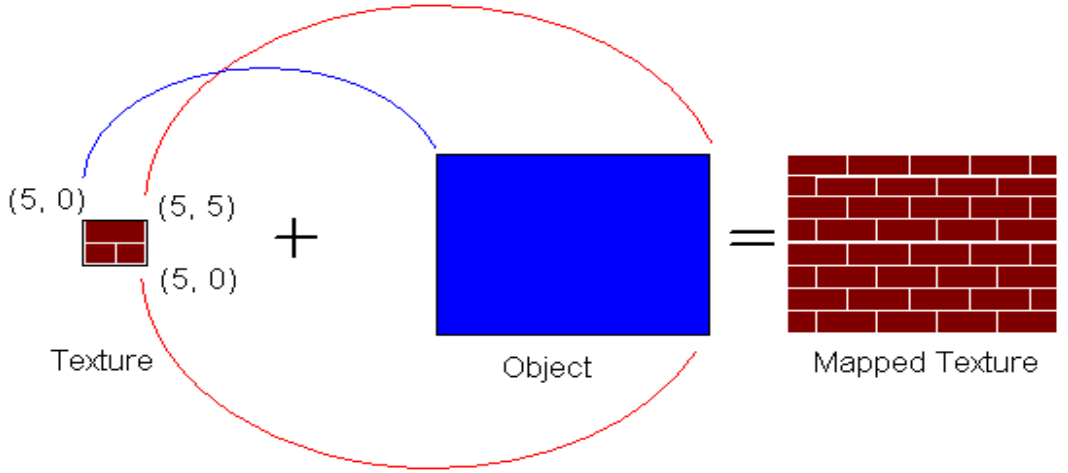
- Tiling allows you to scale repetitive textures to make texture elements just the right size.



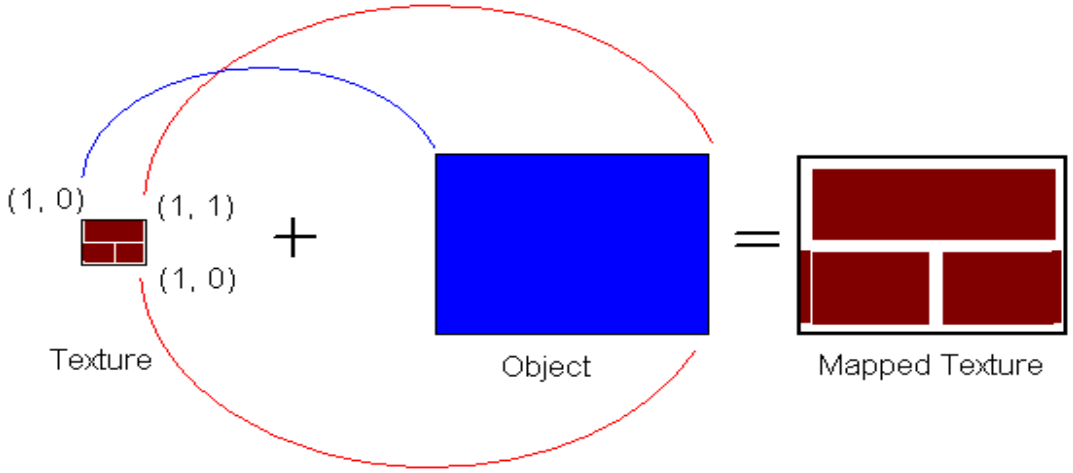
With Tiling

Texture Mapping Style – Tiling: *Example in OpenGL*

glVertex3d (s, s, s)
glTexCoord2d(5, 5);



glVertex3d (s, s, s)
glTexCoord2d(1, 1);



Texture Mapping Style - Stretching

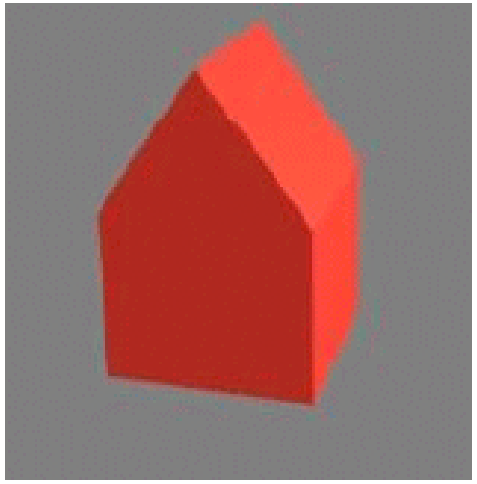
- With non-repetitive textures, we have less flexibility
  - Have to fill an arbitrarily large object with a texture of finite size
  - Can’t tile (will be noticeable), have to stretch instead
  - Example, creating a sky backdrop:

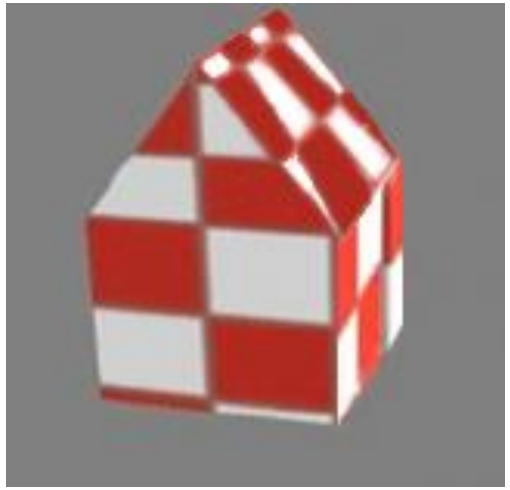
Texture

Applied with stretching

Texture Mapping Complex Geometry

- Sometimes, reducing objects to primitives for texture mapping doesn't achieve the right result.
  - Consider a simple house shape as an example
  - If we texture map it using polygons, we get discontinuities at some edges.

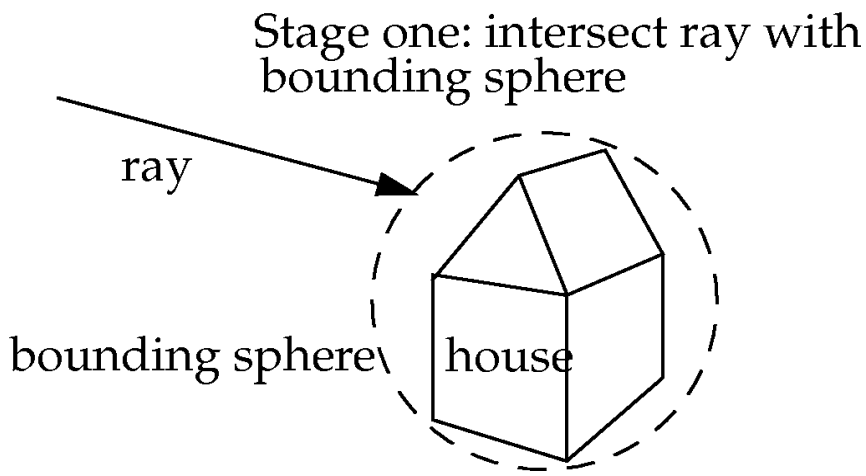


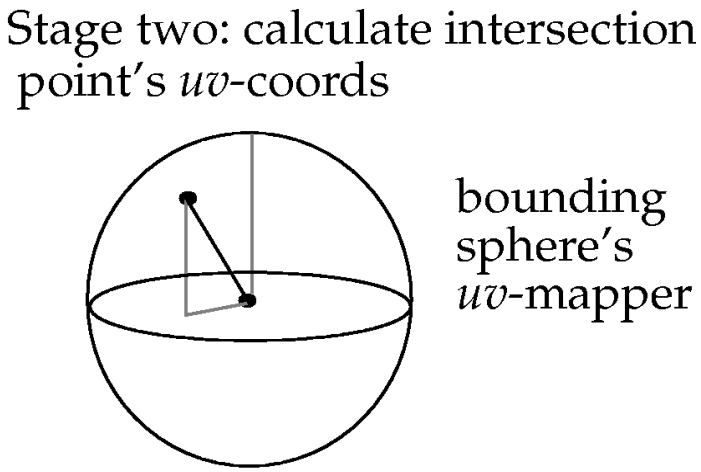


  - Easy solution: Pretend object is a sphere and texture map using the sphere map

Texture Mapping Complex Geometry

- Intuitive approach: Place a bounding sphere around the complex object
  - Find ray's object space intersection with bounding sphere
  - Convert to intersection point *uv*-coordinates

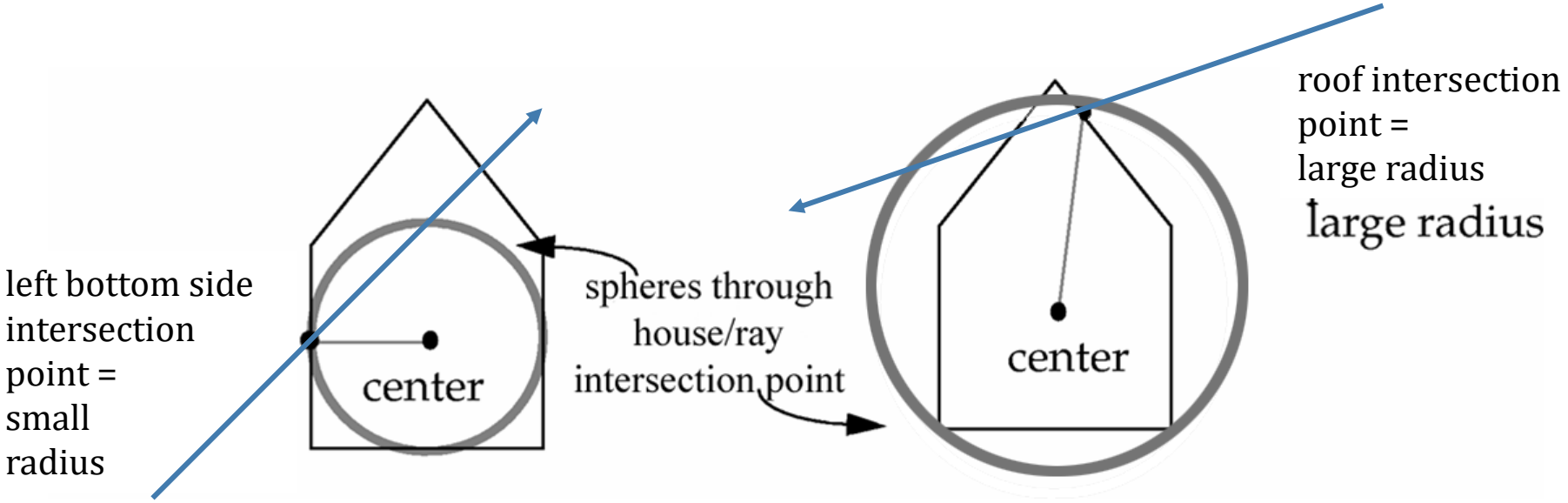




- Don't actually need to construct a bounding sphere!
  - Once have intersection point with object, just treat it as though it were on a sphere passing through point. Same results, but different radii
  - This works because the *(u, v)* coordinates on a sphere don't depend on the sphere's radius – next slide

Texture Mapping Complex Geometry

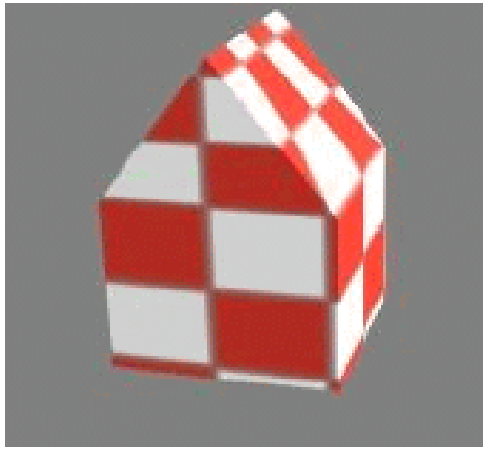
- When we treat the object intersection point as a point on a sphere passing through the point, our “sphere” will vary in radius



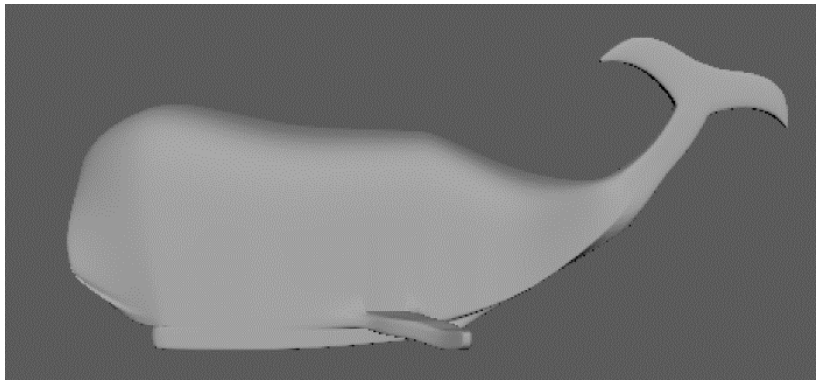
- But radius doesn’t affect (u, v) coordinates on a sphere
  - Only the angles matter (φ and θ in spherical coordinates)

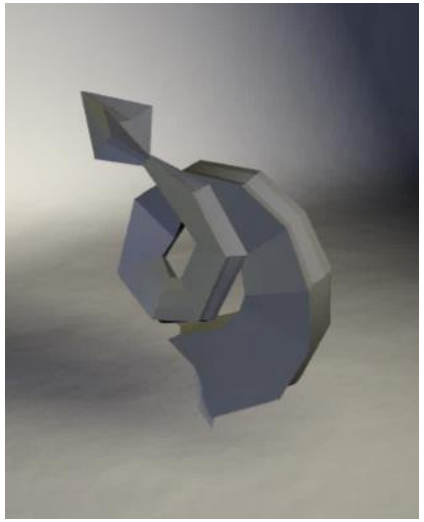
Texture Mapping Complex Geometry

- Results of spherical (u, v) mapping on house:
  - Hey, that looks pretty good. Will it always work?



- For example, what if we want to put a texture on these objects?





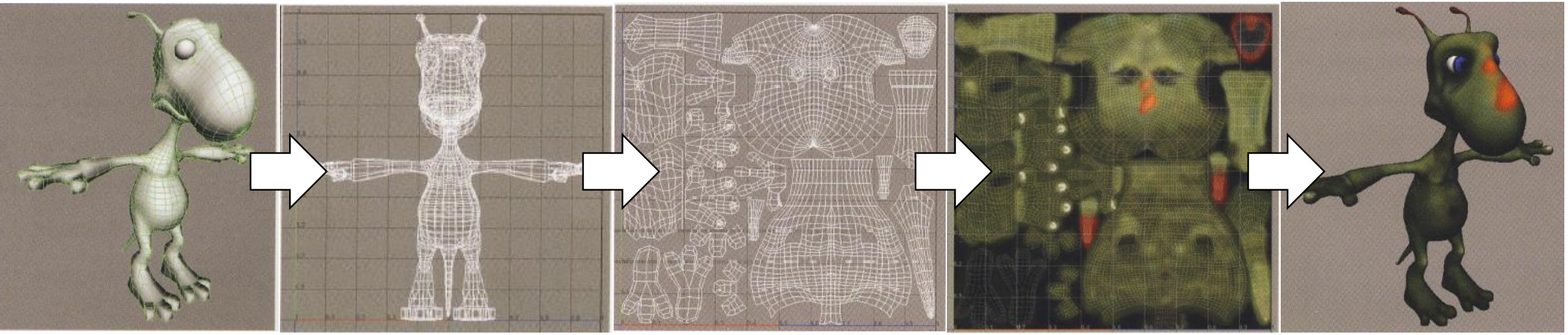
UV Mapping: *Complex Geometry in Real Applications*

- When texture mapping in videogames or films, objects will almost always be more complicated than primitives or that house shape
  - Common objects include humans, monsters, and other organic shapes
- You also want precise control over how the texture map looks on the object
  - Imagine texture mapping a human face with the eyes lined up wrong with the model geometry – viewers would definitely notice!
- Thus, most cases of texture mapping in the "real world" of these industries are done using 3D modelling programs like Maya, Zbrush, Blender, etc.
  - Our examples are from Maya, but the technique would be similar in the other programs

Creating a UV map on Maya
https://youtu.be/HQKDADeuTFc

Complex Geometry in Real Applications

- Here's a very compressed overview of the process:

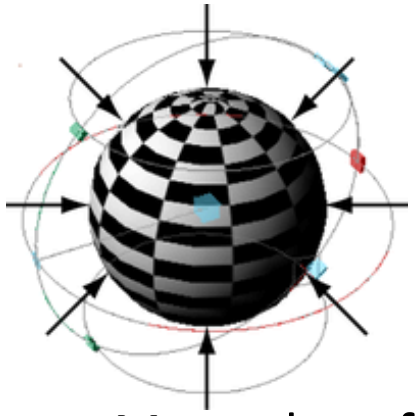


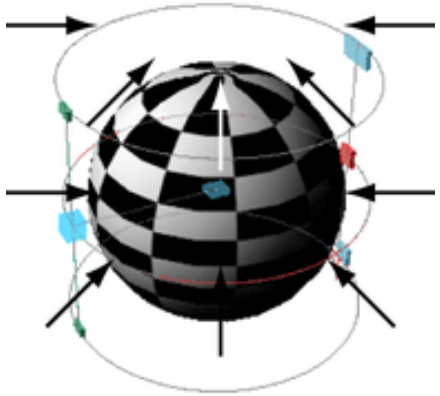
- Ultimately, the goal is to make every face on the object correspond to a section of the (0,0) to (1,1) (*u*,*v*) space

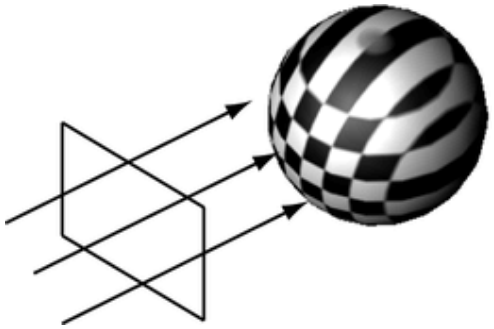
Images from *Learning Maya 7: The Modeling and Animation Handbook*

Complex Geometry in Real Applications

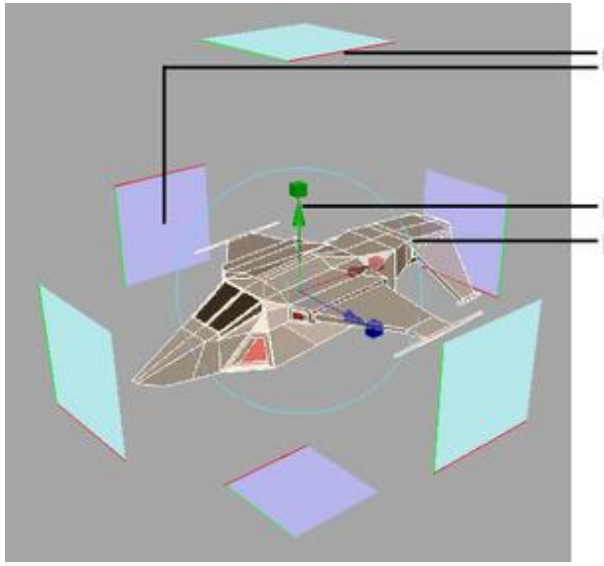
- The main difficulty still lies in generating that mapping
- In addition to the spherical mapping we covered previously (left), in Maya, you can also do cylindrical (middle) or planar mapping (right) when texture mapping objects







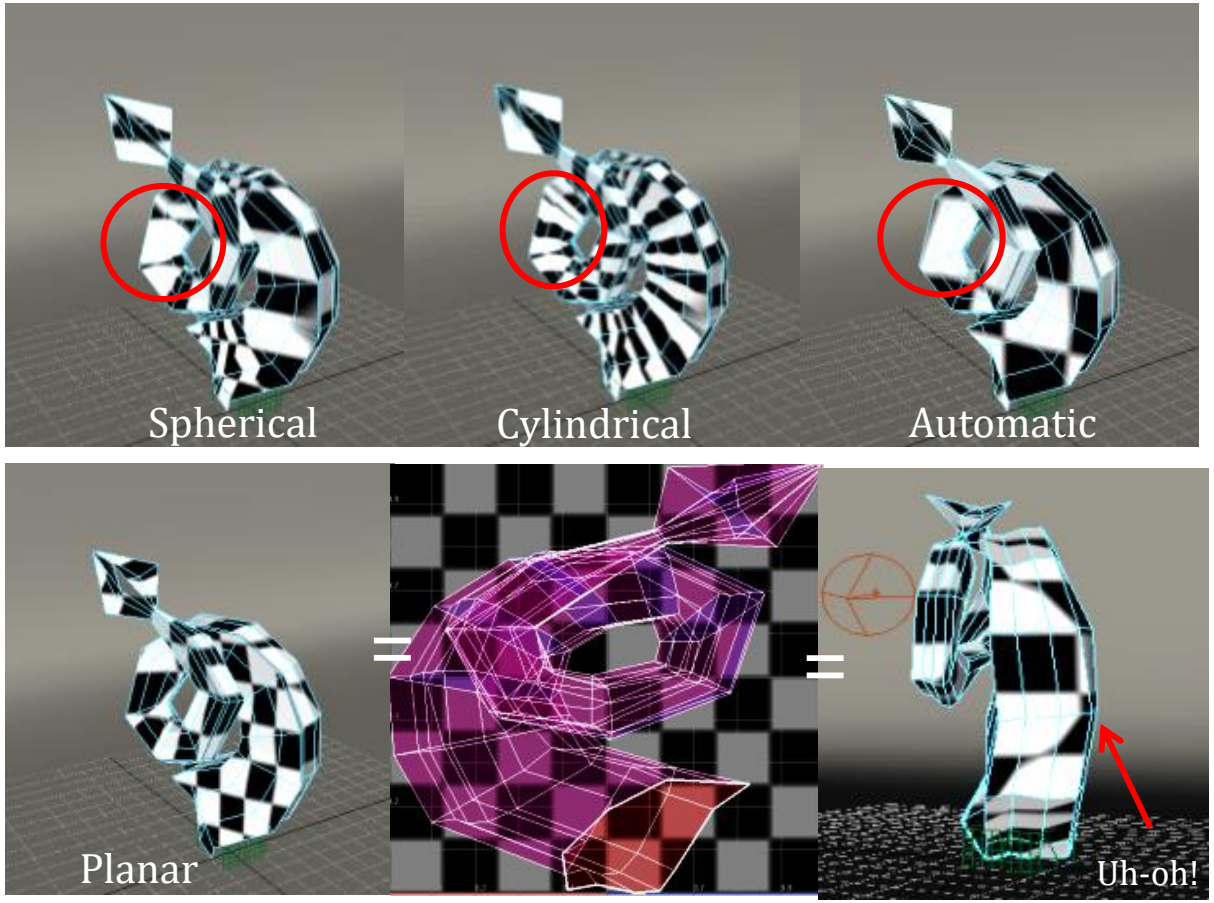
- Maya also offers an “Automatic” mapping
  - Uses multiple projection planes
- Each mapping has drawbacks



Pictures from Autodesk’s *Maya User’s Guide* pages on spherical, cylindrical, planar, and automatic uv mapping.

Complex Geometry in Real Applications

- Testing with a checkerboard pattern is useful when looking for problems with (u, v) mappings.
  - The goal is to minimize uneven distortion of the pattern.
- Spherical, cylindrical, and automatic have a lot of distortion on this twisty object.
  - Red circles show uneven checkers on *all* these mappings – **bad**!
- Planar is okay when viewed from one axis, but the (u, v) map overlaps itself and two axes are ignored.
  - This leads to distortion when viewing from the other axes.

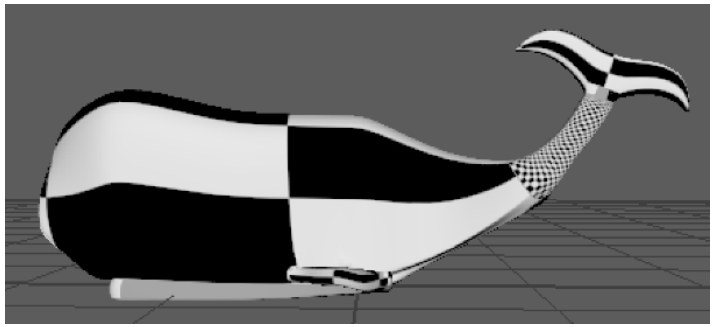


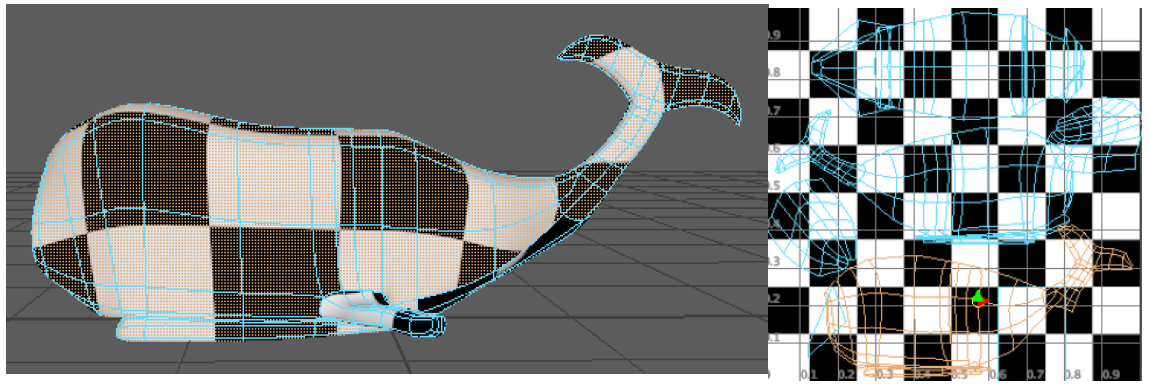
What the planar uv map looks like "unwrapped."
Pink = overlapping squares

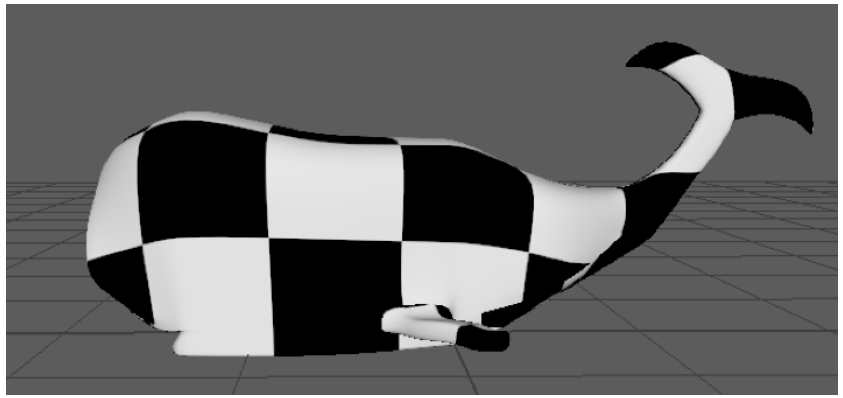
Images by Vivian Morgowicz
Rendered in Maya on the CS125 lighting stage

Complex Geometry in Real Applications

- Maya will give you UV coordinates automatically
  - Most times these UVs aren't quite what we want or look distorted.
- Usually, we need to go in and modify the UVs to get something that we are happy with.
  - We do this in Maya by selecting faces to be part of a UV "shell". We can cut and sew shells as needed.
- Once we get the UV map right, we'll see that the checkerboard is much less distorted
  - It's hard to get the UVs totally perfect. Oftentimes, we can hide some of the problems by putting seams on the bottom or other parts that won't be as visible.



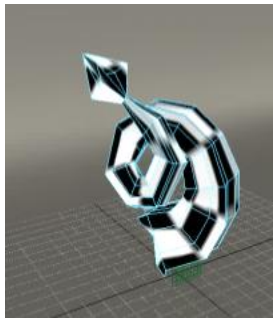




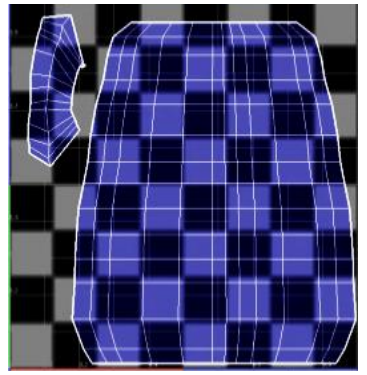
Rendered in Maya using cs125's tutorial

Complex Geometry in Real Applications

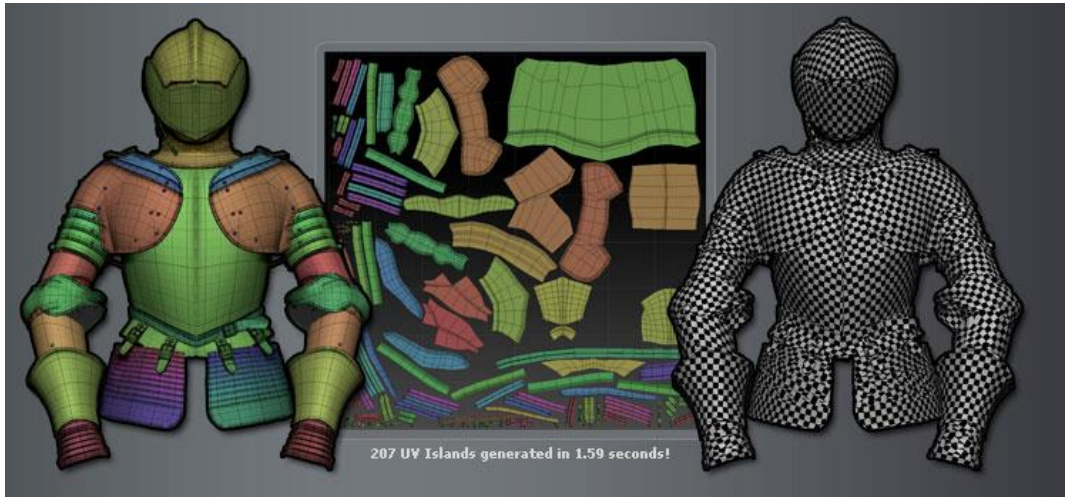
- There is no good solution
- To get the look they want, the modelers will often have to go in and manually cut and sew edges in the (u, v) maps
- However, computers are getting better– there are several complex techniques for making texture maps that look seamless
  - Example: *Seamless Surface Mappings* by Noam Aigerman, Roi Poranne, Yaron Lipman
- Other programs try to generate maps that put the discontinuities in places where the real objects would have seams.

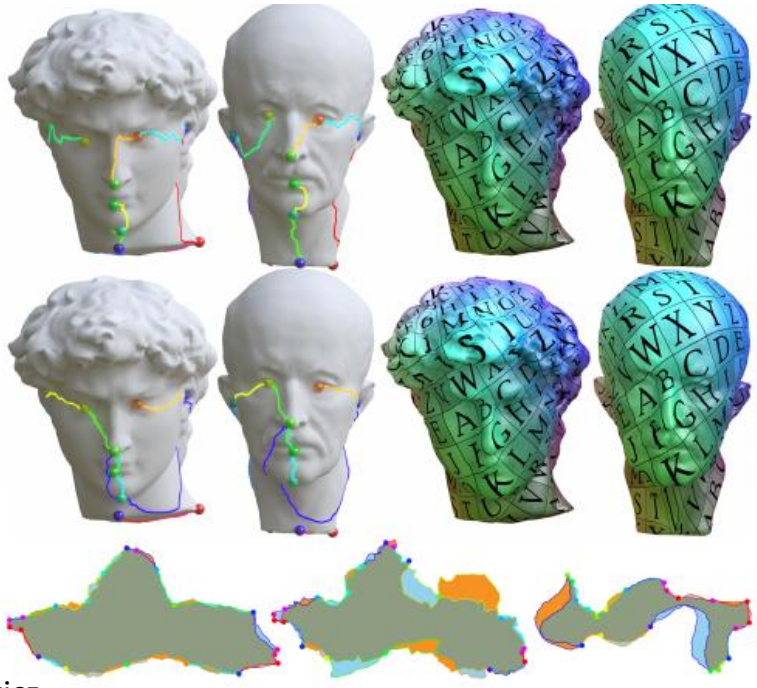


=



Handmade





Top Images by Vivian Morgowicz
Left image from http://pixologic.com/zbrush/features/UV-Master/

Other Forms of Texture Mapping

- Bump Mapping
- Displacement Mapping
- Environment Mapping

Surface Detail

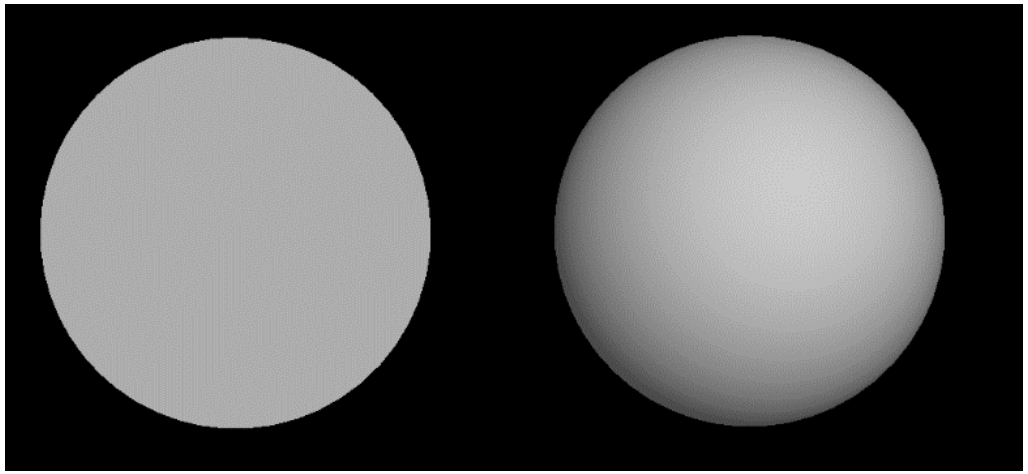
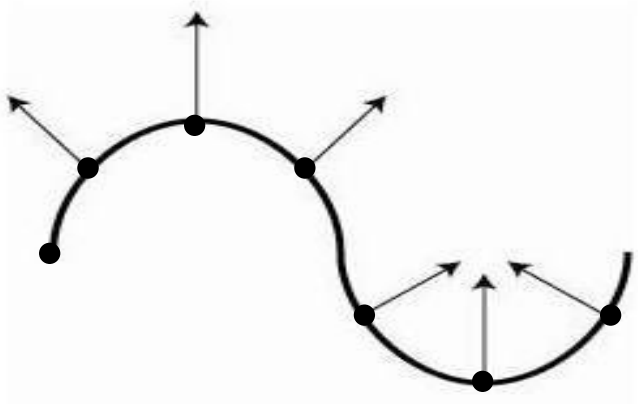


Image credit: Dave Kilian, ‘13

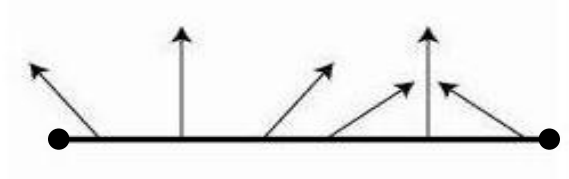
- Observation
  - What if we replaced the 3D sphere on the right with a 2D circle?
  - The circle would have fewer triangles (thus renders faster)
  - If we kept the sphere’s **normals**, the circle would still look like a sphere!
  - Works because human visual system infers shape from patterns of light and dark regions (“shape from shading”). Brightness at any point is determined by normal vector, not by actual geometry of model

Idea: Surface Detail

- Two ideas: use texture map either to vary normals or vary mesh geometry itself. First let's vary normals
- Start with a high-poly (high polygon count) model
- Encode high-poly normal information into texture
- *Decimate* the mesh (remove triangles)
- Map texture (i.e., normals) onto low-poly mesh
- In fragment shader:
  - Look up normal for each fragment in normal texture map and perform lighting equation as per usual



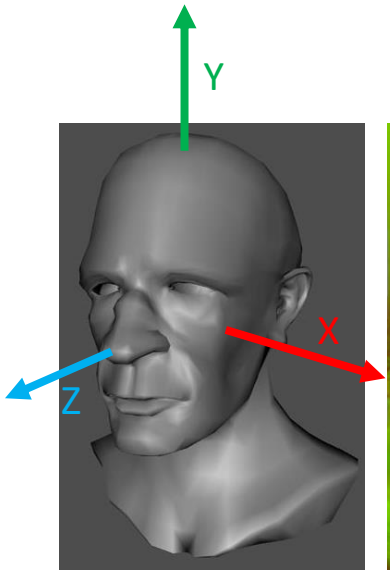
Original high-poly model



Low-poly model with high-poly model's normals preserved

Normal Mapping

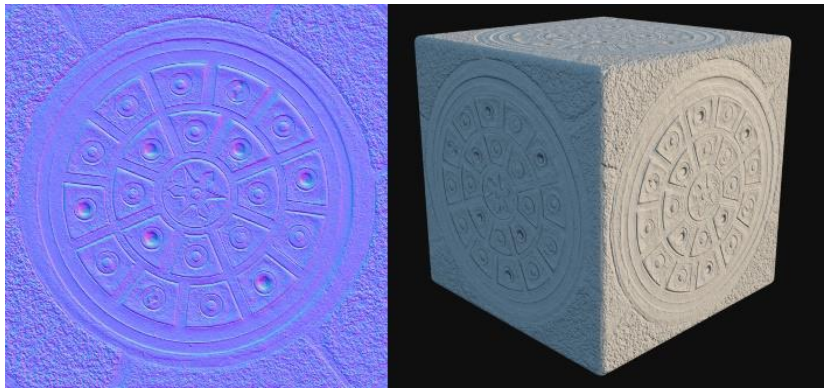
- Idea: Fill a texture map with normals
  - Fragment shader samples 2D color texture, then interprets it as normals
- Easiest to render, hardest to produce from scratch
- Types
  - Object space normal map
    - In the xyz object-space coordinate system, interprets R as N_x ; G as N_y ; B as N_z
  - Tangent space normal map
    - In the uvw object-space coordinate system, interprets R as N_u ; G as N_v ; B as N_w



Original mesh


Object-space normal map

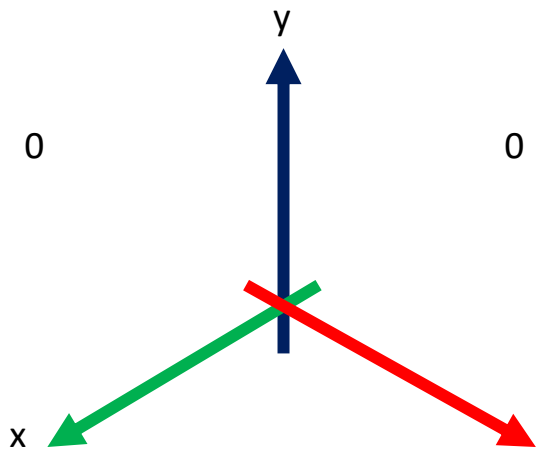

Mesh with normal map applied (colors still there for visualization purposes)

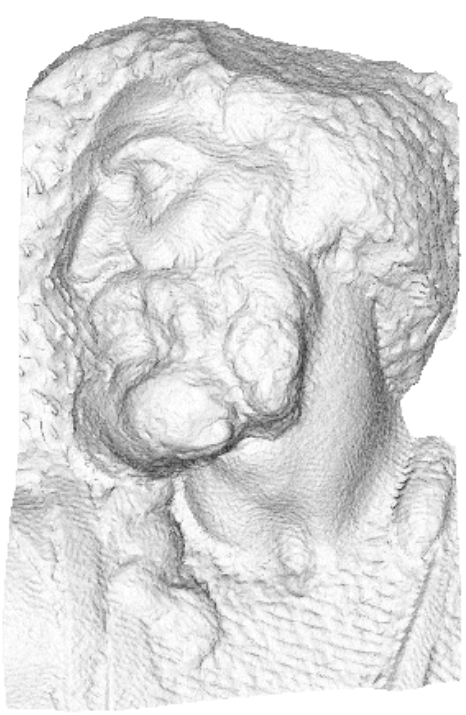

Tangent space normal map

http://www.3dvf.com/forum/3dvf/Blender-2/modelisation-organique-tutoriel-sujet_16_1.htm
(right click on Google Chrome. Translate to English)

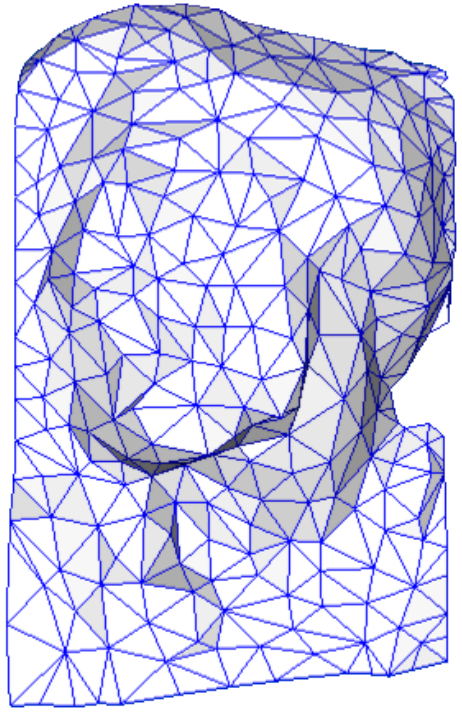
Normal Mapping: *Example*

- Normal mapping can completely alter the perceived geometry of a model

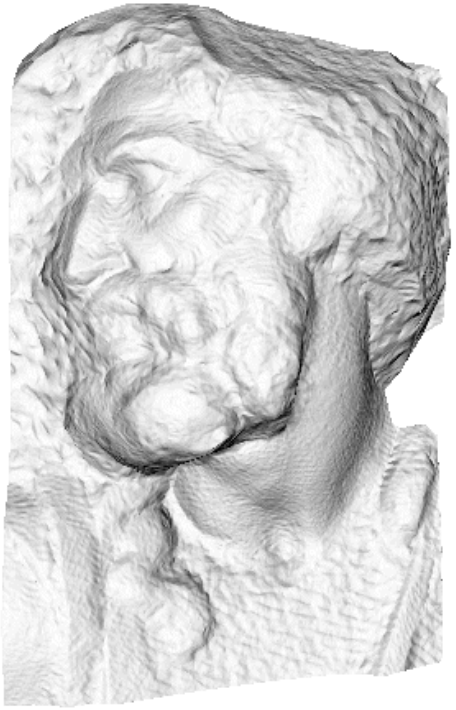




original mesh
4M triangles



simplified mesh
500 triangles

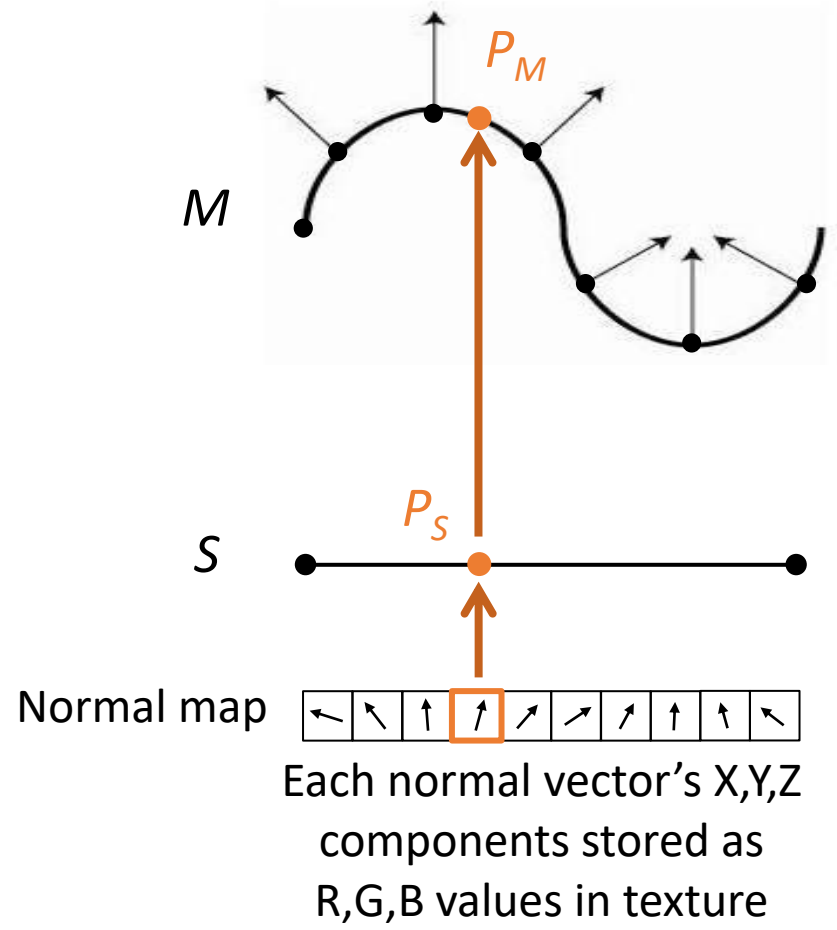


simplified mesh
and normal mapping
500 triangles

Image courtesy of www.anticz.com

Creating Normal Maps

- Algorithm ("baking the texture map")
  - Original mesh *M* simplified to mesh *S*
  - For each pixel in normal map for *S*:
    - Find corresponding point on *S*: P_S
    - Find closest point on original mesh: P_M
    - Average nearby normals in *M* and save value in normal map
- Manual (Sculpting)
  - Artist starts with *S*
  - Uses specialized tools (e.g. Zbrush) to draw directly onto normal map
- Photoshop feature
  - https://youtu.be/1PmMWdwZAvg



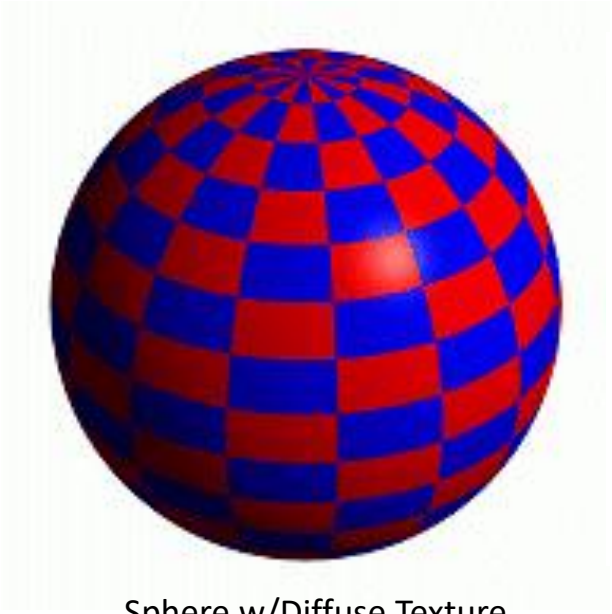
Normal Mapping: *Videos*

- Difference between object space and tangent space normal maps (0:32 to 1:16): https://youtu.be/m-6Yu-nTbUU?t=32

- Finding tangent space basis vectors (6:22 to 7:10): https://youtu.be/6_-NNKc4lrk?t=382

Bump Mapping

- Another Way to Perturb Normals
- Use textures to alter the surface normal
  - Does not change the actual shape of the surface
  - Just shaded as if it were a different shape

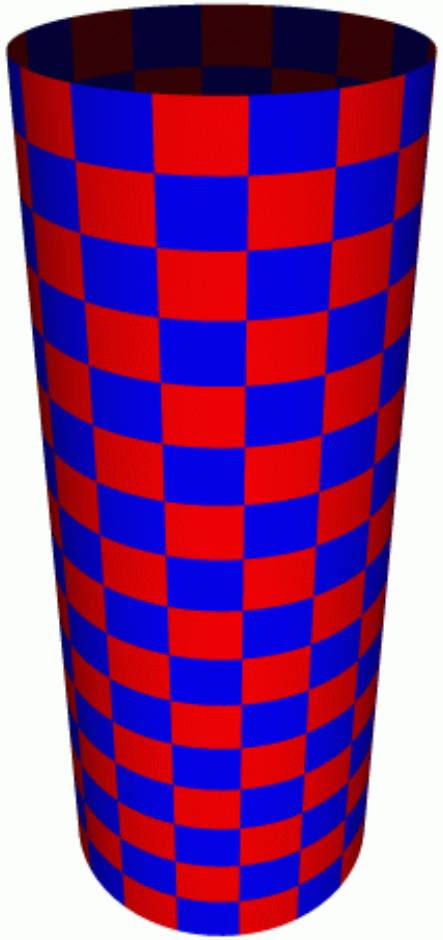


Sphere w/Diffuse Texture

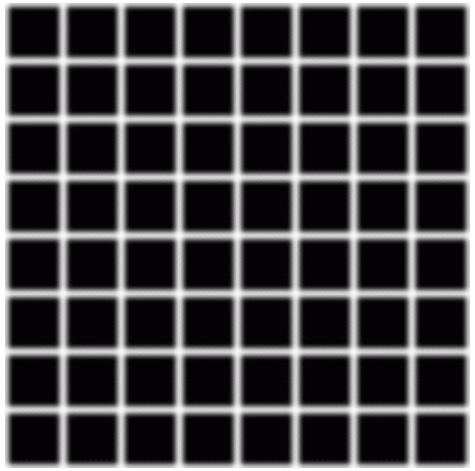
Swirly Bump Map

Sphere w/Diffuse Texture & Bump Map

Another Bump Map Example



Cylinder w/Diffuse Texture Map

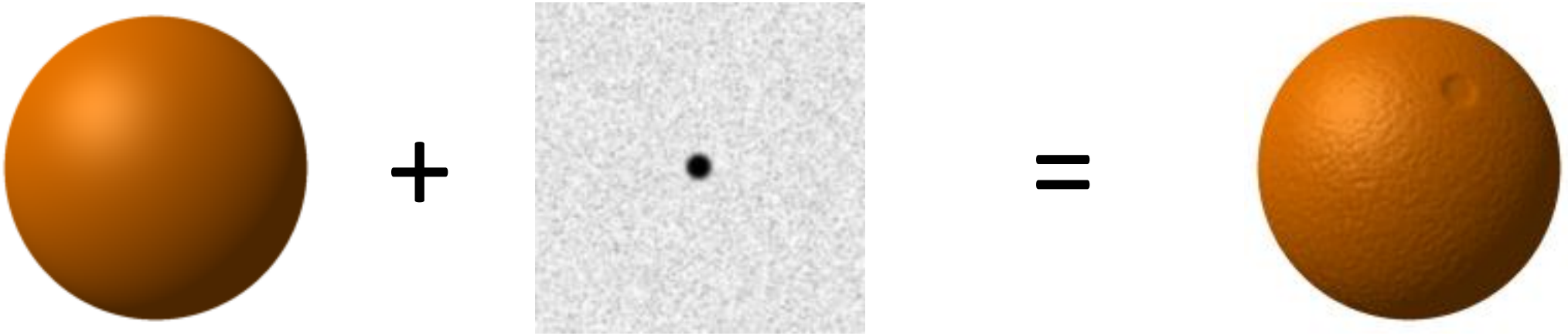


Bump Map

Cylinder w/Texture Map & Bump Map

Bump Mapping: Example

So, we can see in the image below that the black dot creates a little “indent” in the sphere, even though we haven’t changed the actual surface geometry



Original object (plain sphere)

Bump map

Reminder :

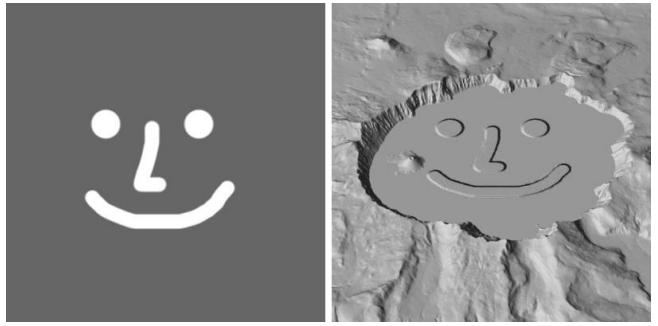
- Black is the minimum height w/ respect to surface and white
- White is the maximum height delta

Sphere with bump-mapped normal (Single bump map wrapped around entire sphere)

http://cse.csusb.edu/tongyu/courses/cs520/notes/texture.php

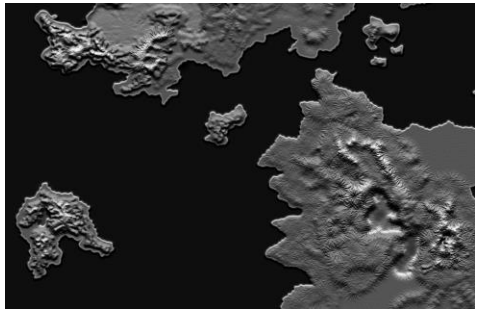
Creating Bump Maps

- Custom bump maps are pretty intuitive – paint the texture black for lower bumps and white for higher bumps
- An easier alternative: find a texture that you want to simulate with bump maps and then convert it to grayscale
  - This works pretty well for some pictures (like the terrain)
  - Doesn't work for brick walls (notice how the cement lining is considered the highest points?)
    - Solution: invert the highlights and shadows in an image-editing software

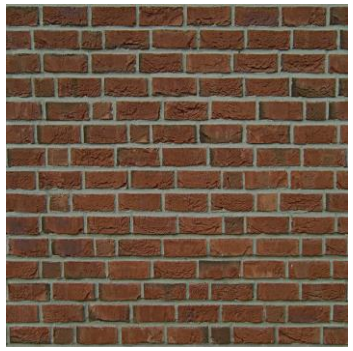


A custom bump map, can be created using brushes in Photoshop

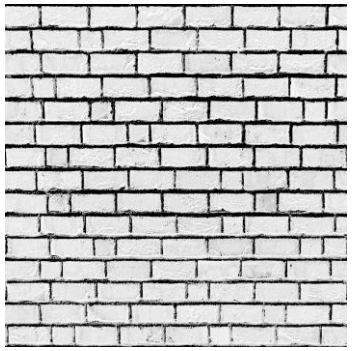
Original terrain image



Terrain image converted to grayscale



Brick texture converted to grayscale

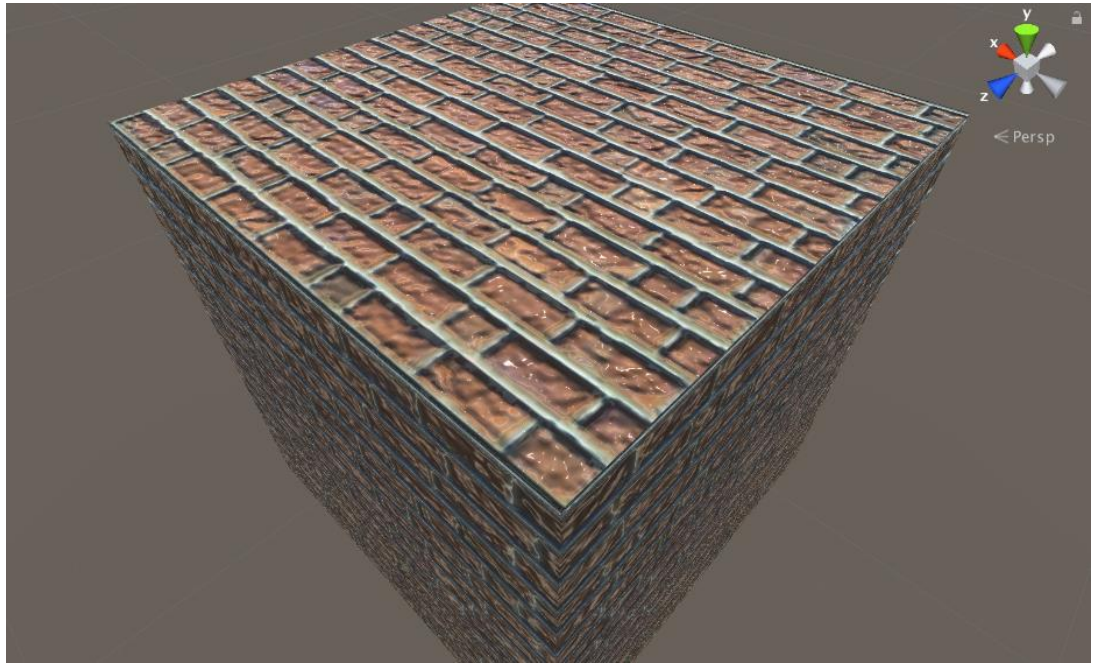


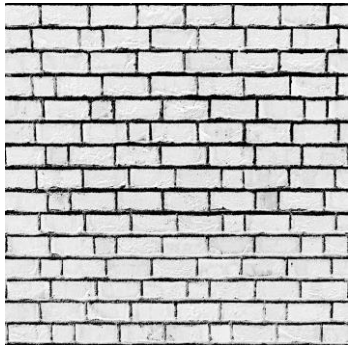
Grayscale image with inverted highlights and shadows

[Custom bump map source](#) / [Terrain image](#) / [Brick bump maps](#)

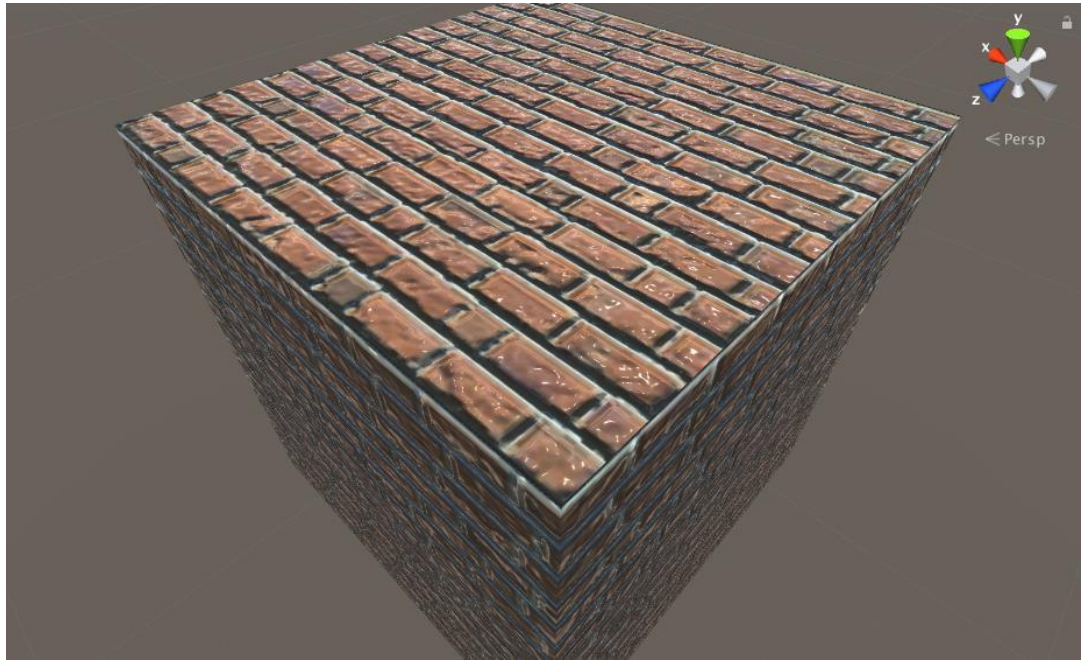
Examples of Using the Brick Bump Maps

Using the simple grayscale texture to bump map
See how the cement lining looks raised?





Using the inverted grayscale image
Now our bricks are higher than our cement lining!

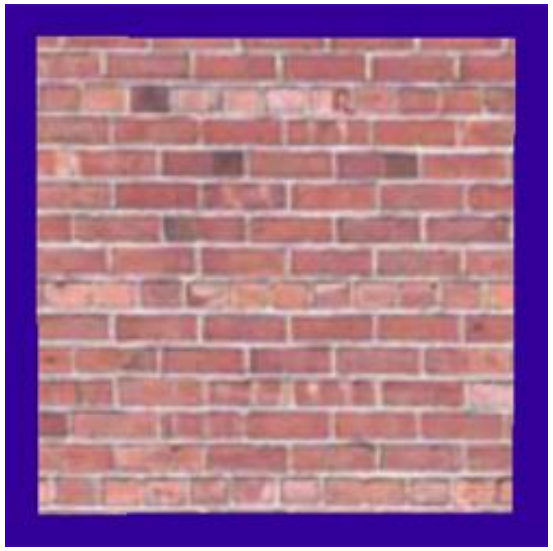


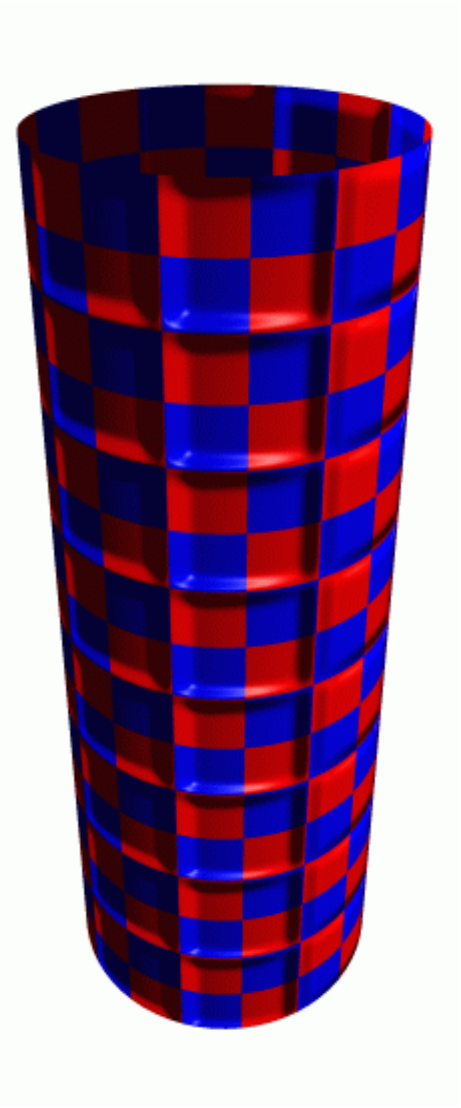
Images created in Unity!

What's Missing?

- There are no bumps on the silhouette of a bump-mapped object

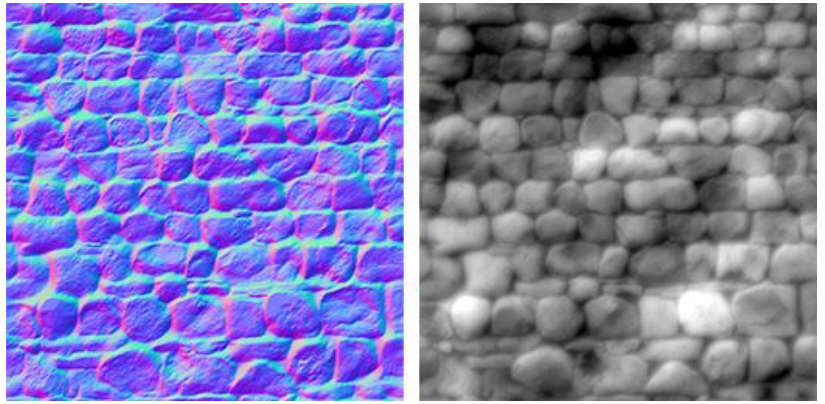
- Bump maps don’t allow self-occlusion or self-shadowing



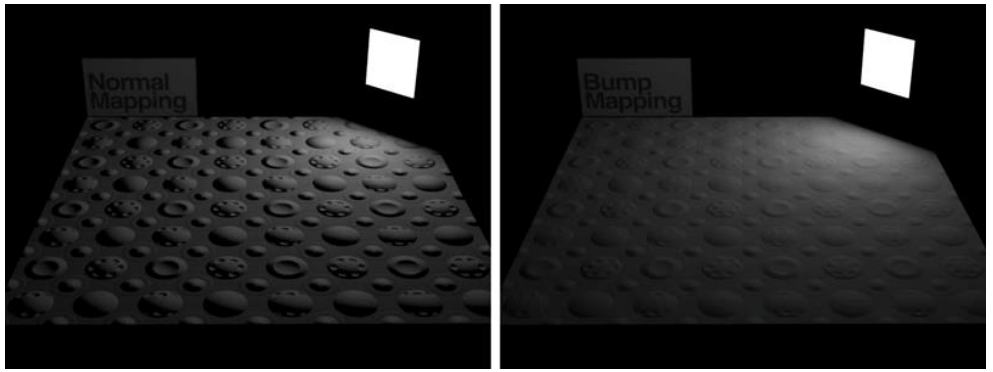


Normal Mapping vs Bump Mapping

- Normal mapping is sampling a texel from a normal map and converting the (R, G, B) value to a (N_x, N_y, N_z) normal (for object space) or (N_u, N_v, N_w) (for tangent space)
- Bump mapping is sampling a texel from a bump map and converting the intensity value to its relative height value, then using the neighboring height values to calculate the tangent space normal of an object at the given point
- Pros to bump mapping:
  - Easier to create custom bump maps
- Cons:
  - Normal maps require less computation to convert the texels to normal vectors
  - Normal maps have more predictable results (and are thus more favored by artists)



On the left, a normal map for bump mapping a stone wall. On the right, a height map for bump mapping a stone wall.

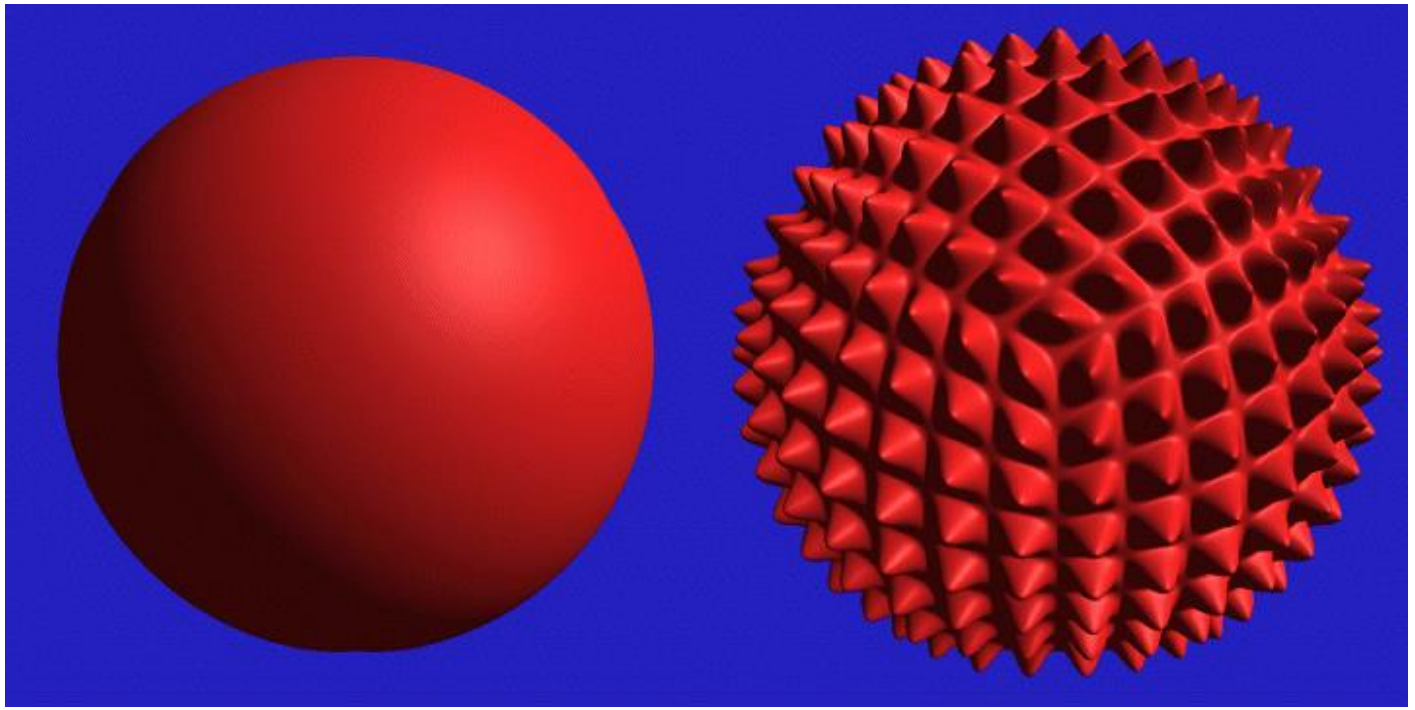


Normal map (left) / Bump map (right)
Bump map has less pronounced normals because each normal is the average of its surrounding

Stone wall bump map source

Other Techniques: *Displacement Mapping*

- Use the texture map to actually move the surface point
- The geometry must be displaced before visibility is determined



Other Techniques: *Displacement Mapping*

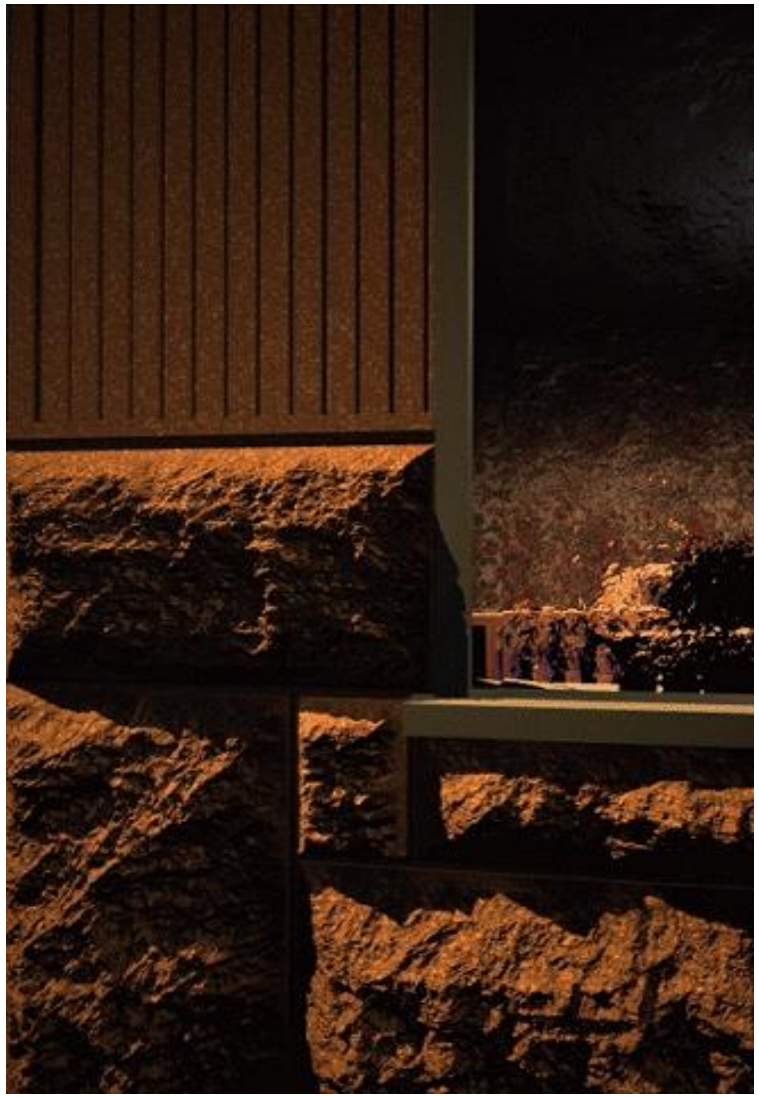


Image from:

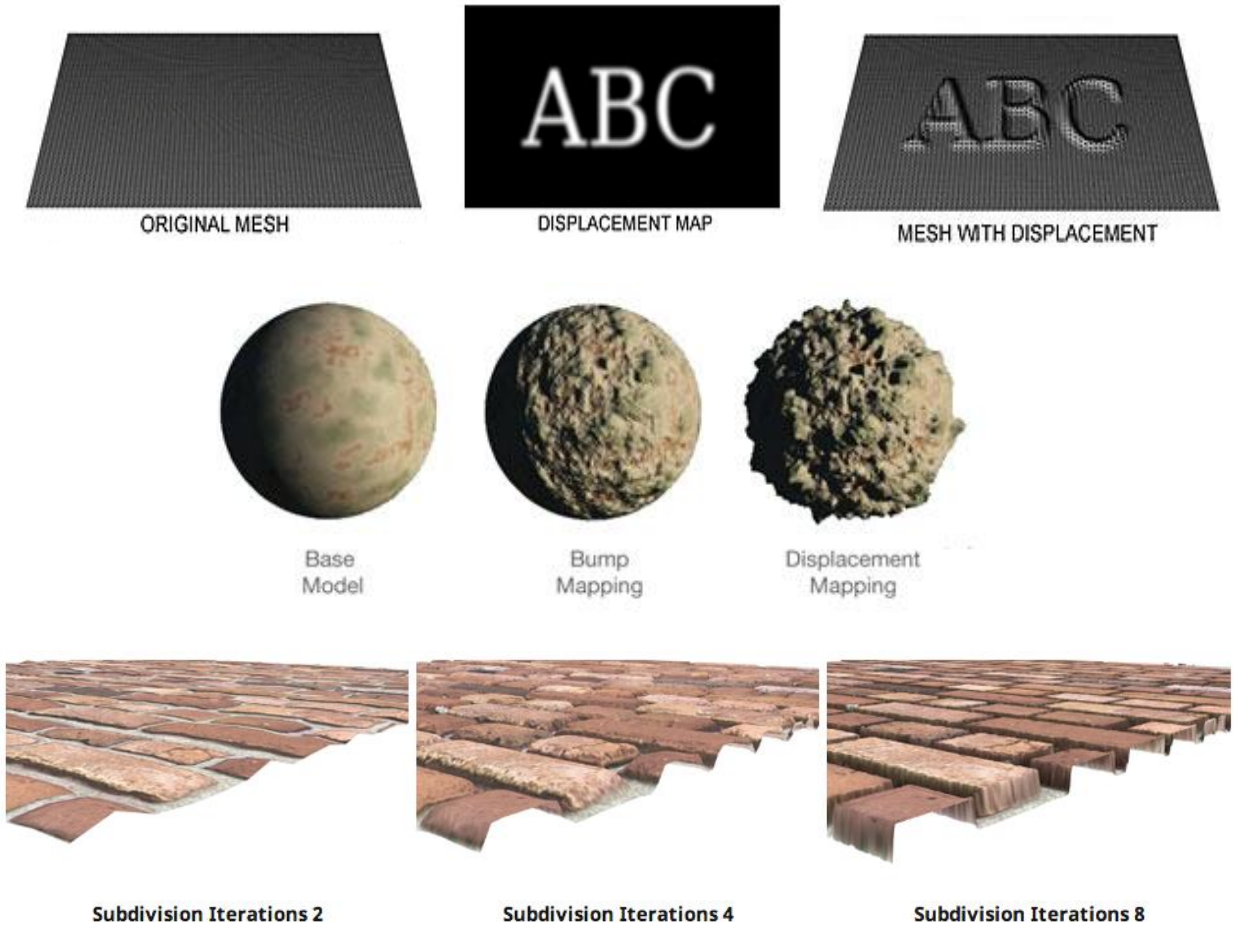
Geometry Caching for Ray-Tracing Displacement Maps

by Matt Pharr and Pat Hanrahan.

Note the detailed shadows cast by the stones

Other Techniques: *Displacement Mapping*

- Actually *move the vertices* along their normals by looking up height deltas in a height map
  - Displacing the vertices of the mesh will deform the mesh, producing different vertex normals because the face normals will be different
  - Unlike bump/normal mapping, this will produce correct **silhouettes** and **self-shadowing**
- By default, does not provide detail *between* vertices like normal/bump mapping
  - To increase detail level we can subdivide the original mesh
  - Can become very costly since it creates additional vertices



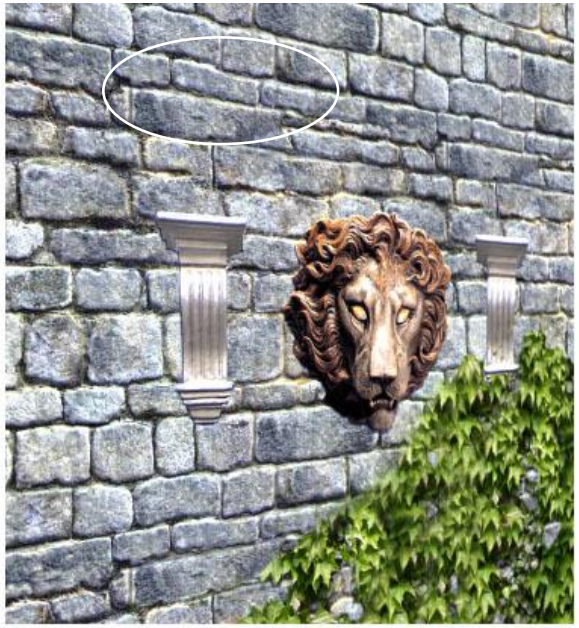
Displacement map on a plane at different levels of subdivision

http://en.wikipedia.org/wiki/Displacement_mapping
http://www.nvidia.com/object/tessellation.html
https://support.solidangle.com/display/AFMUG/Displacement

Comparison

Texture Mapped

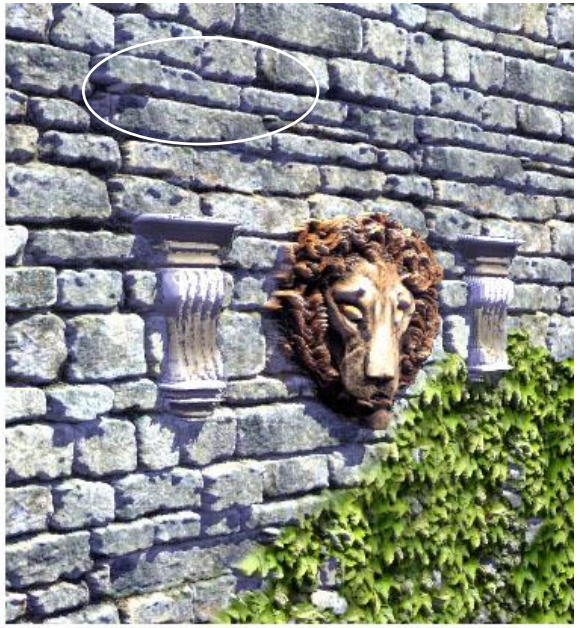
Creases between bricks look flat



Normal Mapped

Parallax Mapped

Creases look progressively deeper

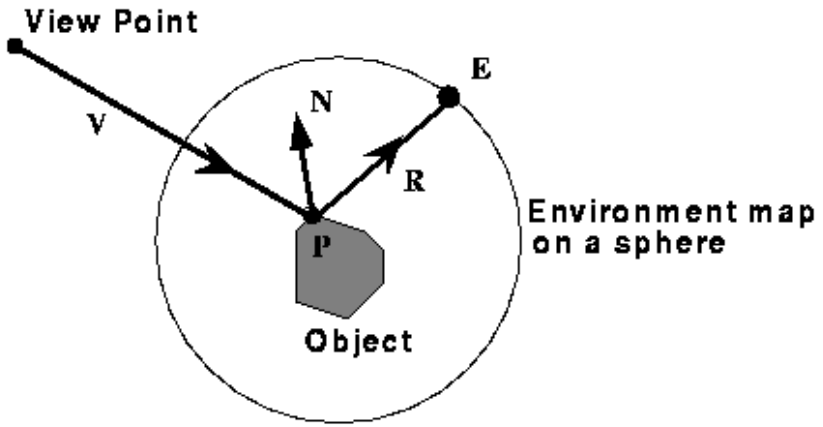


Steep Parallax Mapped

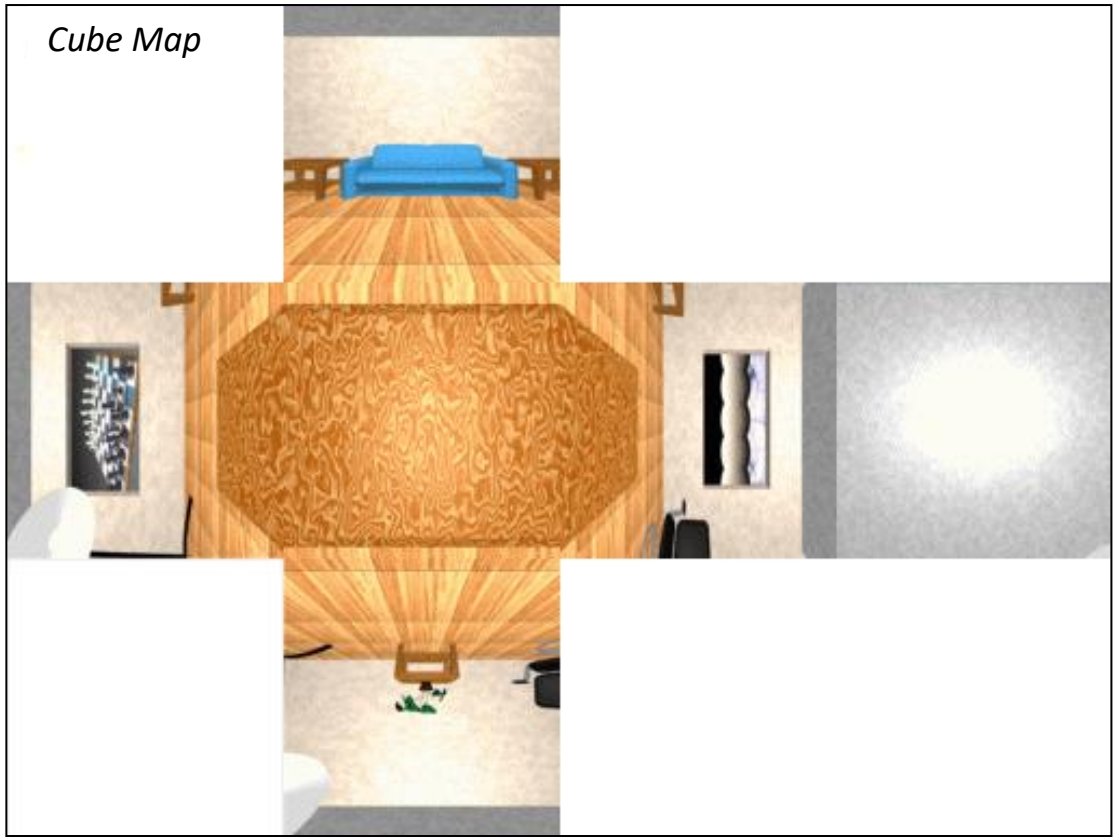
Objects have self-shadowing

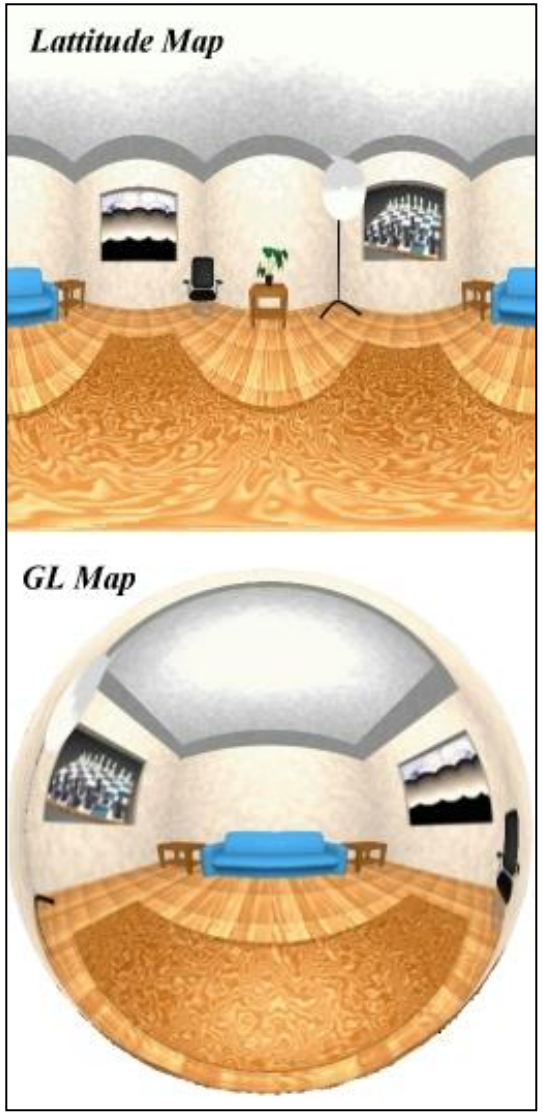
Environment Maps

- **Environment Maps** is an efficient image-based lighting technique for approximating the appearance of a reflective surface by means of a precomputed texture image. The texture is used to store the image of the distant environment surrounding the rendered object.
  - We can simulate reflections by using the direction of the reflected ray to index a spherical texture map at "infinity".
  - Assumes that all reflected rays begin from the same point.

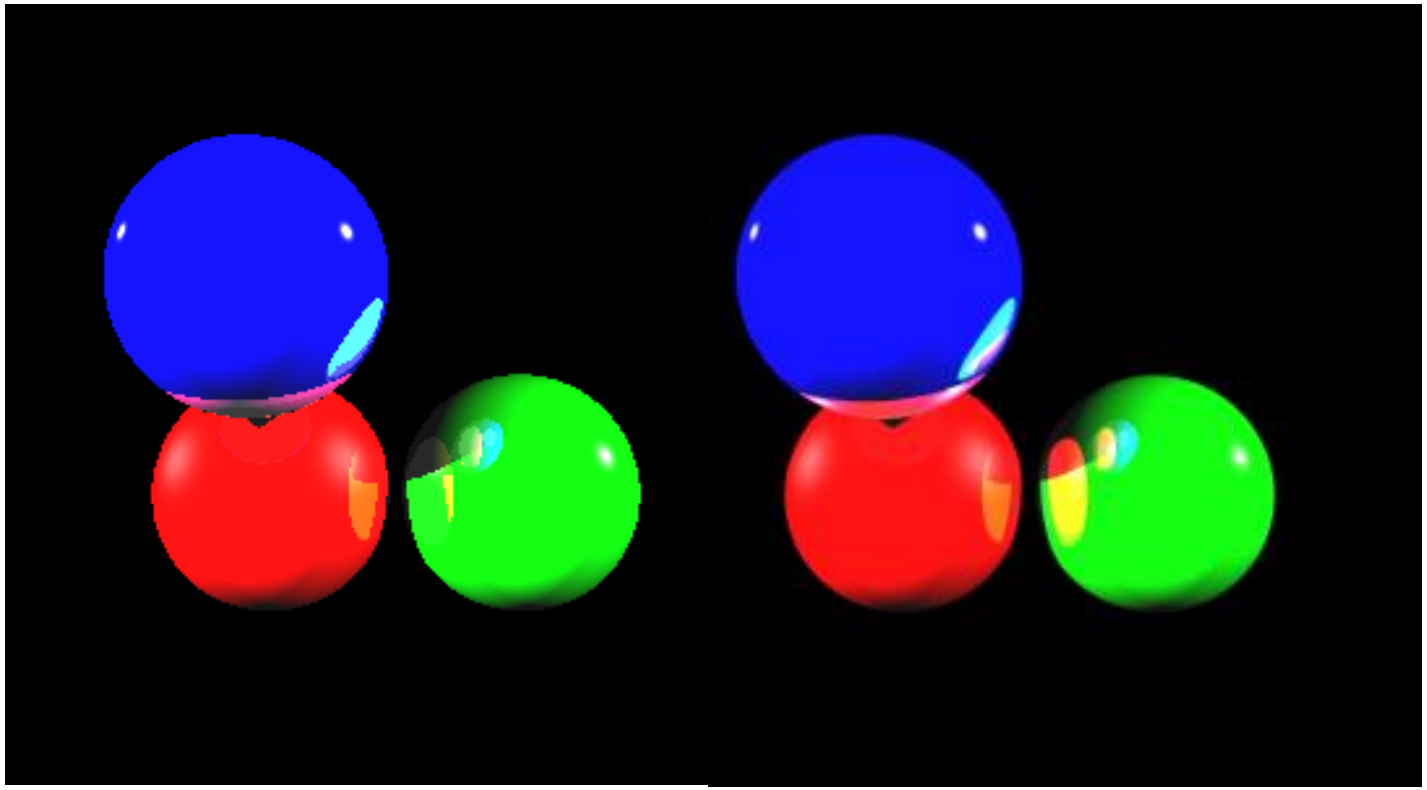


What's the Best Layout?

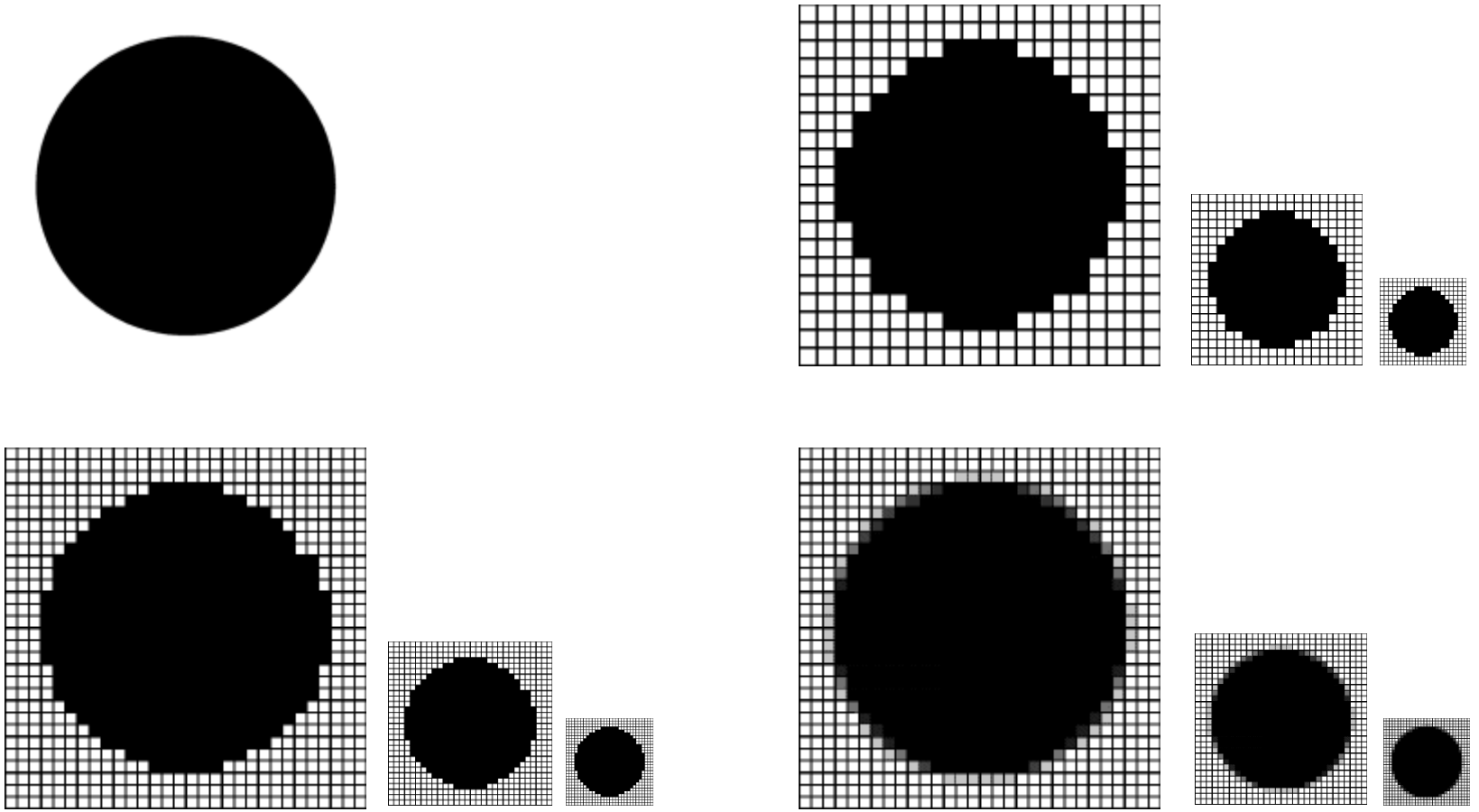




Aliasing - Antialiasing



Ταύτιση – Αντιταύτιση (Antialiasing)



Antialiasing

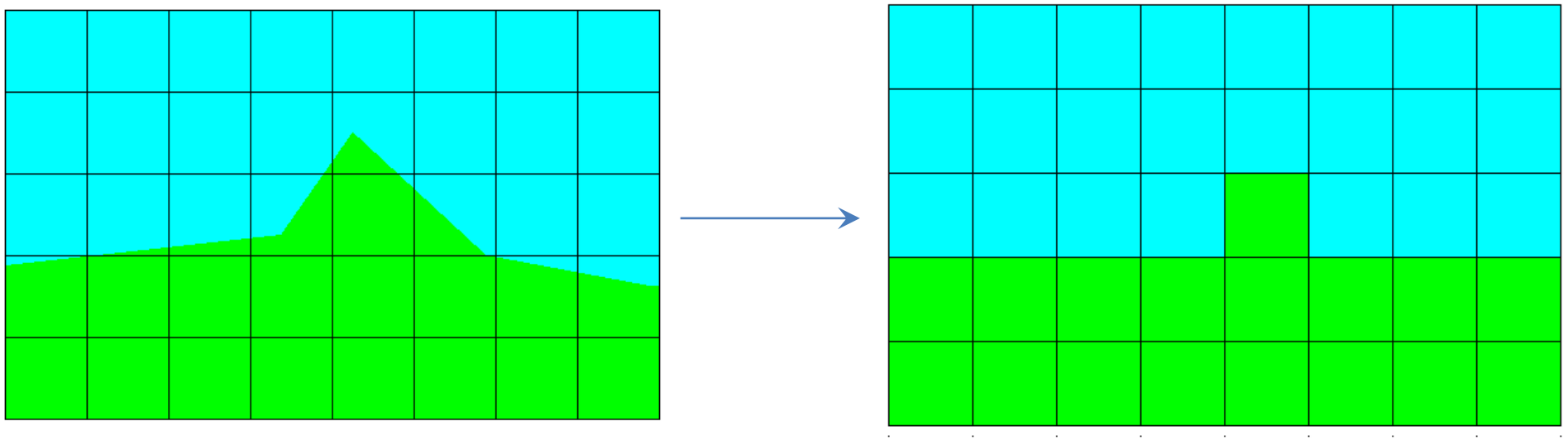
- Aliasing: distortion artifacts produced when representing a high-resolution signal at a lower resolution.
- Anti-aliasing : techniques to remove aliasing

Aliased polygons (jagged edges)

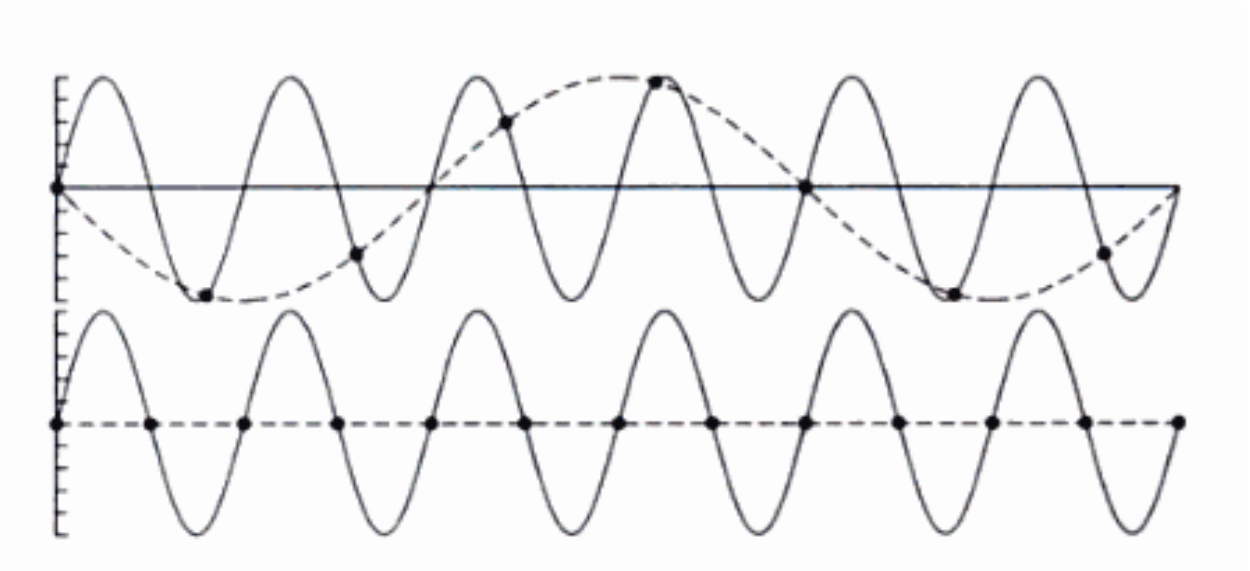
Anti-aliased polygons

Why Does Aliasing Happen?

- Sampling frequency is too low with respect to the signal frequency
- In the example below, the pixel size is too large compared to the original scene



Nyquist Limit



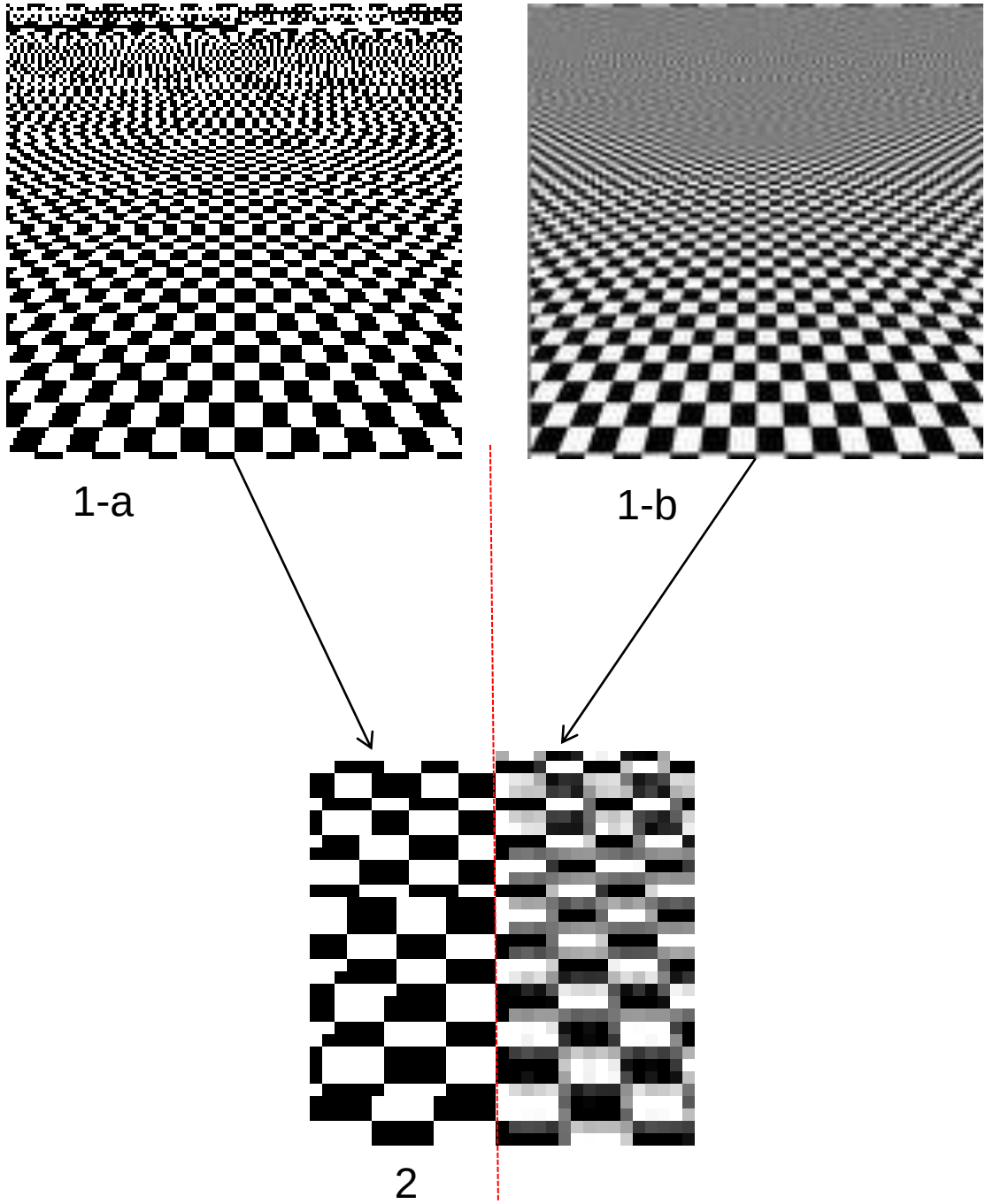
- The signal frequency (f_{signal}) should be no greater than half the sample frequency (f_{sample})
- $f_{signal} < 0.5\ f_{sample}$
- In the top, $f_{signal} = 0.8\ f_{sample}$ -> cannot reconstruct the original signal
- In the bottom $f_{signal} = 0.5\ f_{sample}$ -> the original signal can be reconstructed by slightly increasing the sampling rate

Wagon-wheel Effect (temporal aliasing)

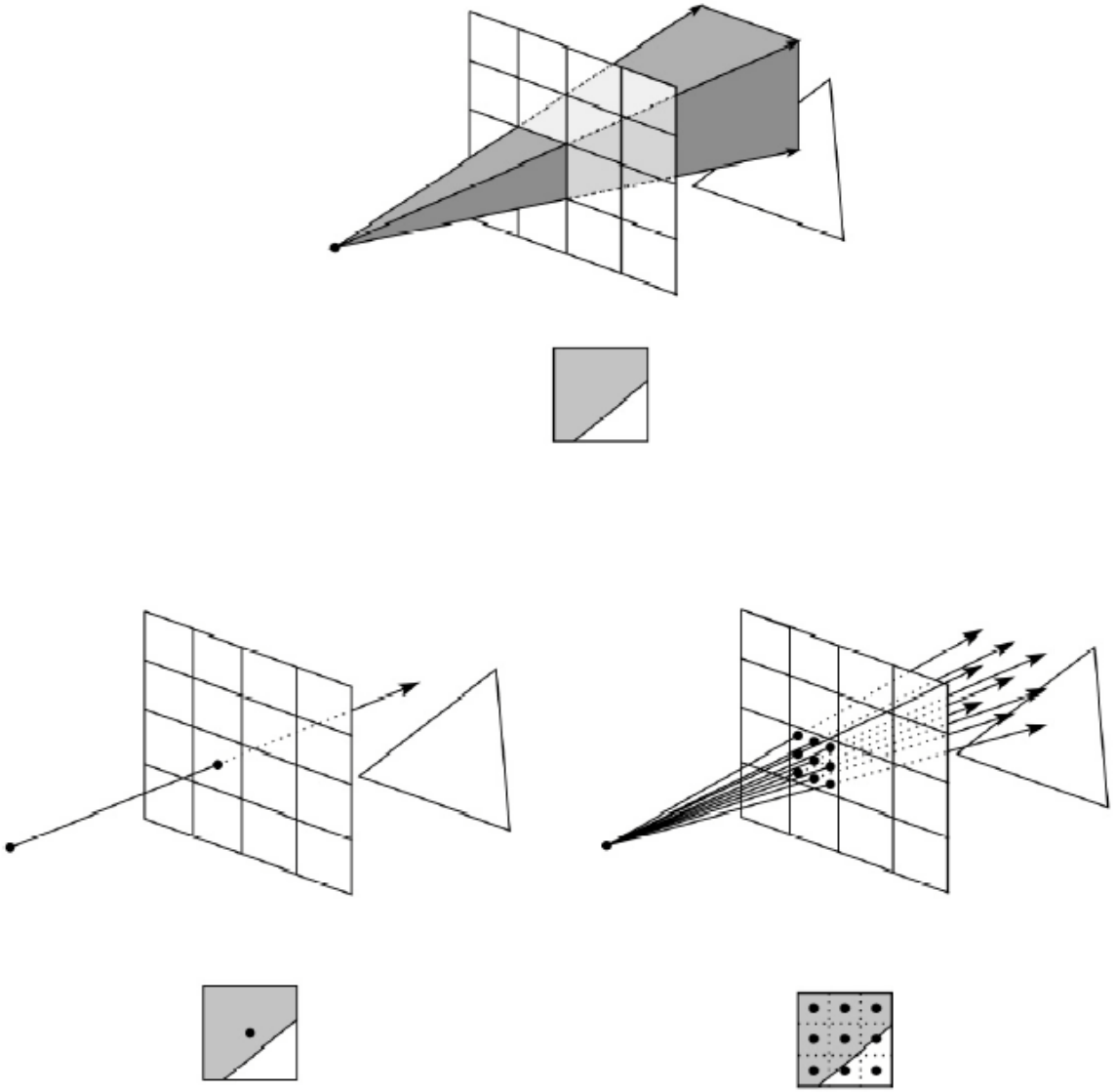
Antialiasing

- Fig 1-a, near the top
  - checkerboard is very distant, the image is impossible to recognize, and is not aesthetically appealing.

- Fig 1-b is anti-aliased,
  - near the **top**: checkerboard **blends into gray**, which is usually the desired effect when the resolution is insufficient to show the detail.
  - near the **bottom**: the edges appear **much smoother** in the anti-aliased image.

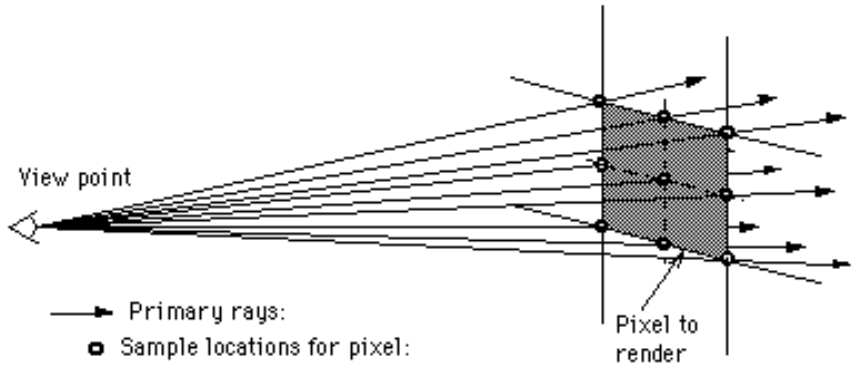


Antialiasing in ray tracing

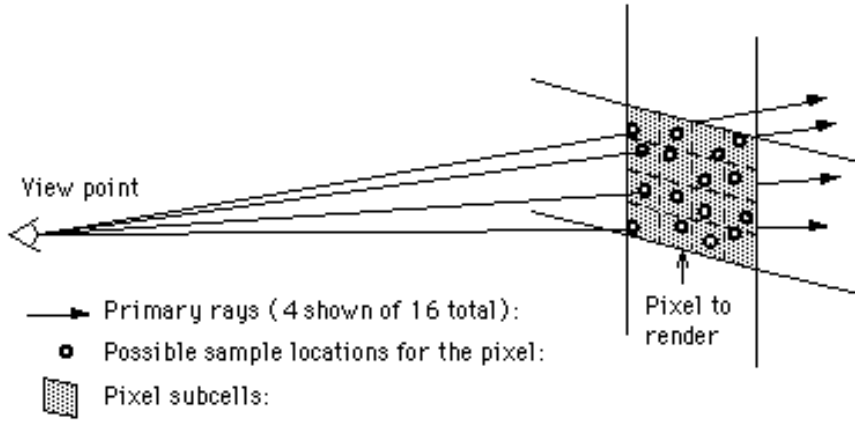


Antialiasing in ray tracing

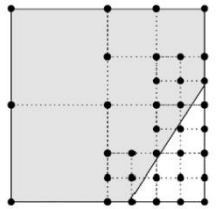
- **Supersampling**



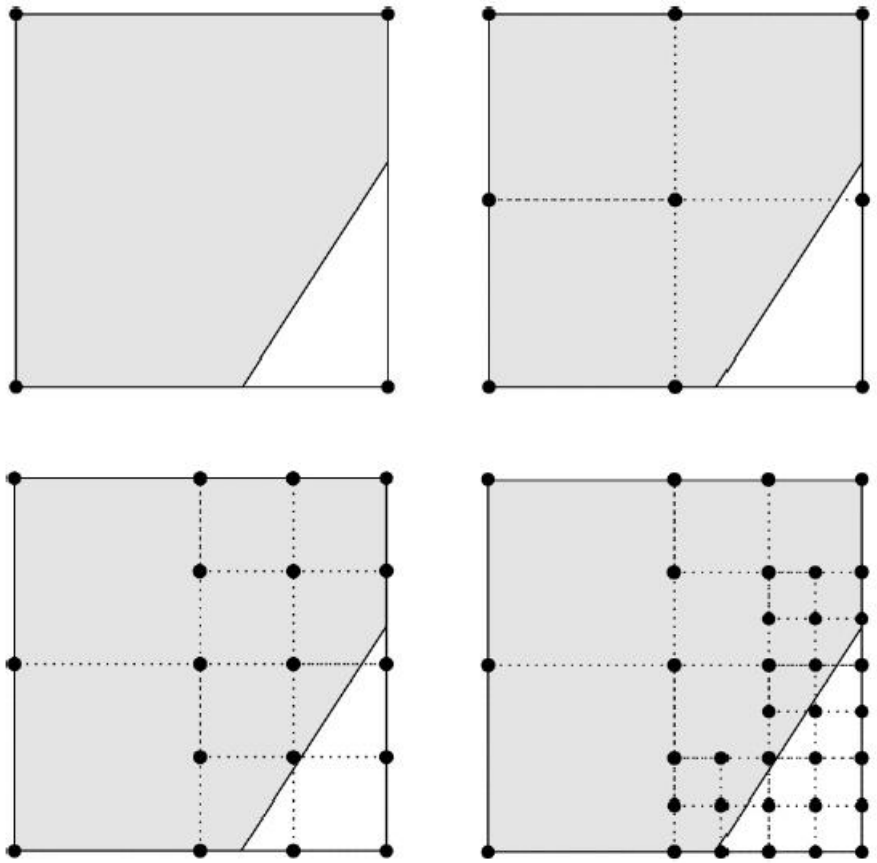
- **Stochastic Sampling**



- **Adaptive Sampling**

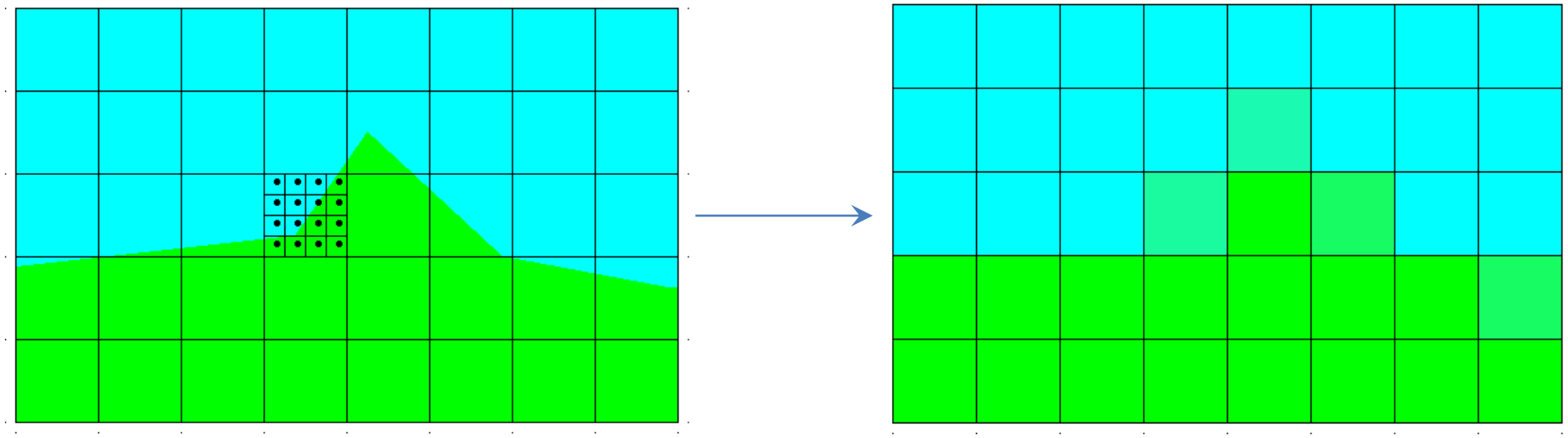


Adaptive sampling

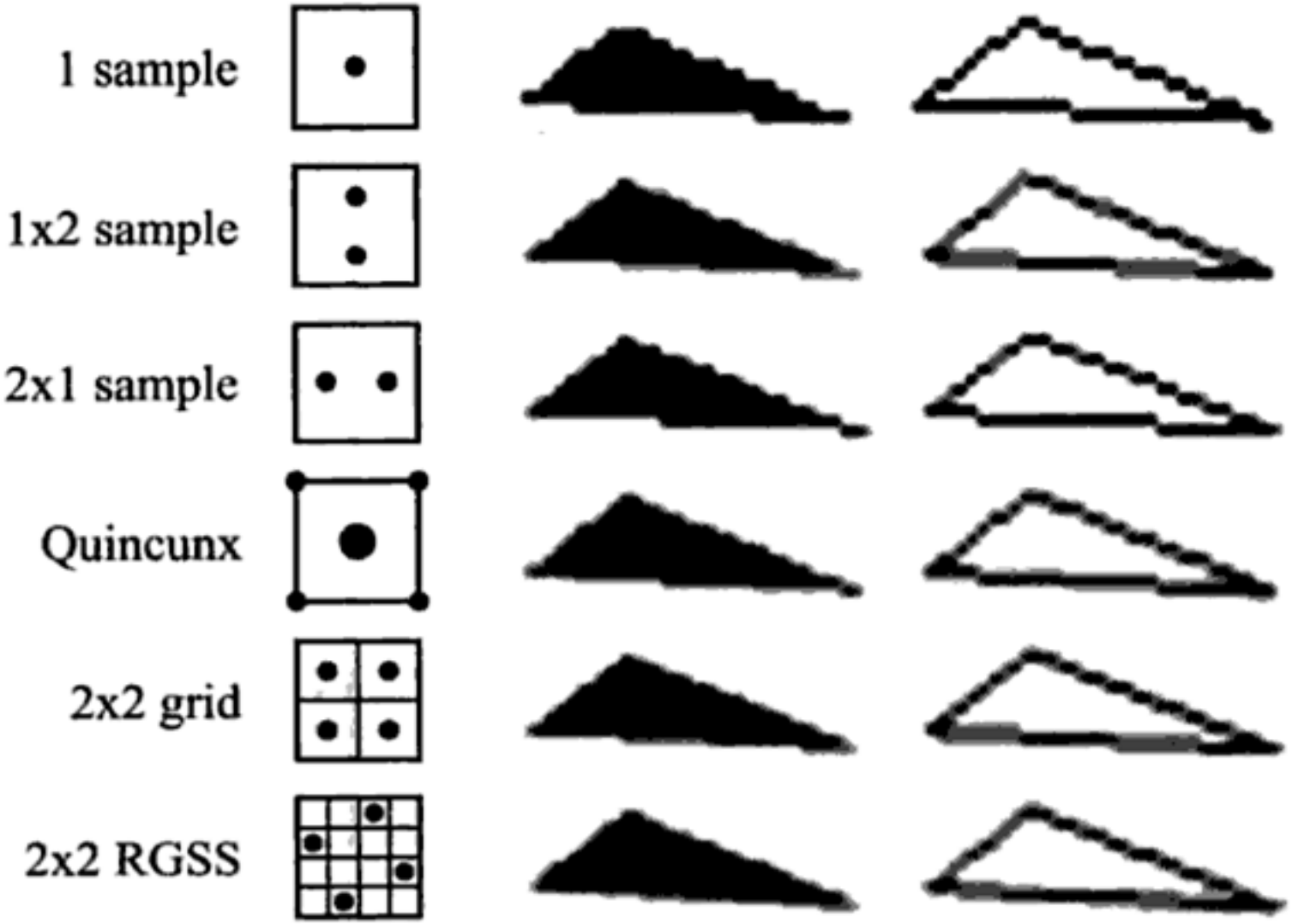


Anti-aliasing by Subsampling

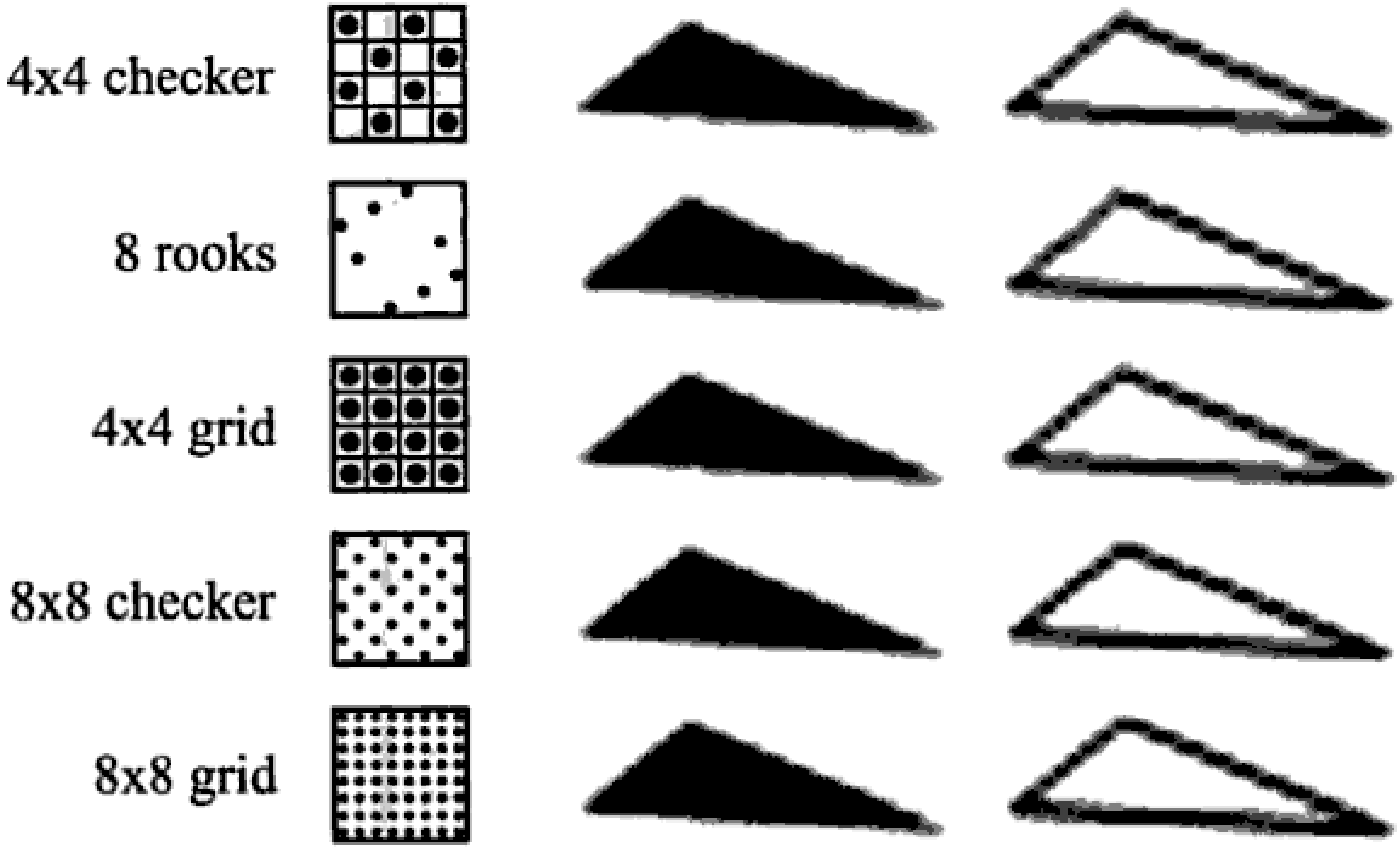
- Each pixel is subdivided (sub-sampled) into n regions
- Obtain the color at each subpixel
- Compute the average color



Different Sampling Schemes

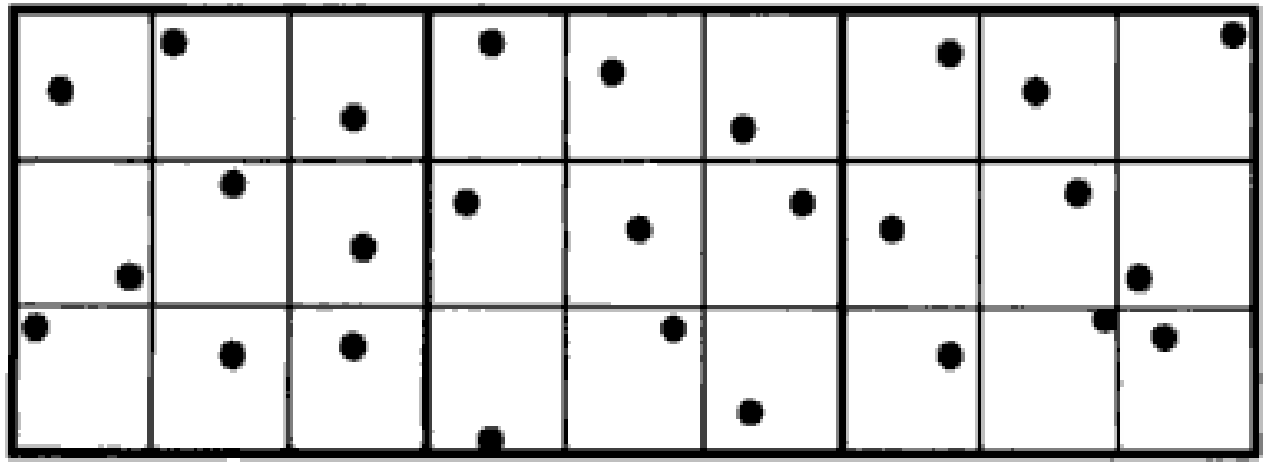


Different Sampling Schemes



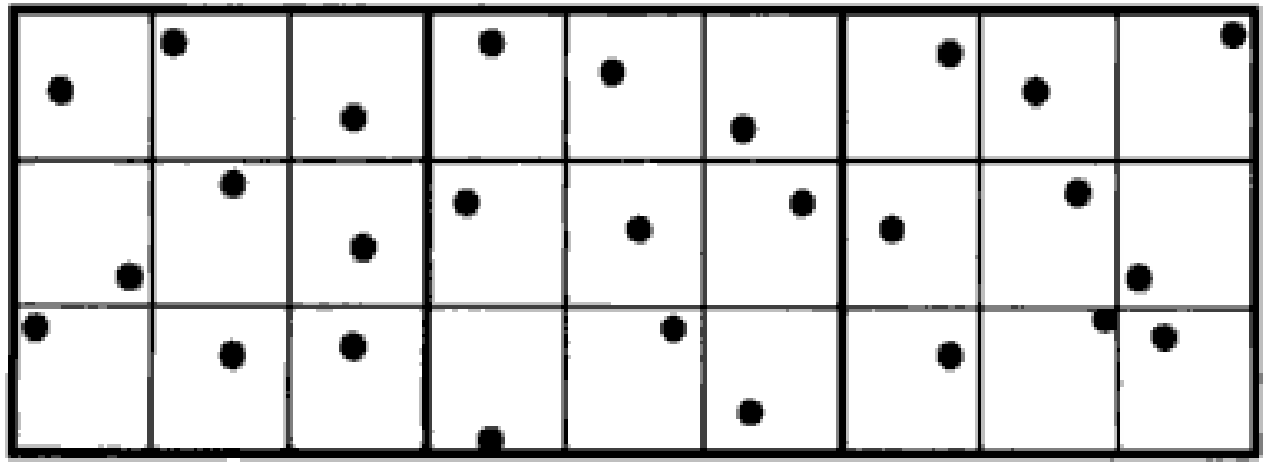
Stochastic Sampling

- A scene can be produced of objects that are arbitrarily small
- A regular pattern of sampling will exhibit some sort of aliasing
- By irregular sampling, the higher frequencies appear in the image as noise rather than aliases
- Humans are more sensitive to aliases than noise



Stochastic Sampling

- One approach to solve this is to randomly sample over the pixels
- Jittering : subdivide into n regions of equal size and randomly sample inside each region
- Compute the colour at the sample and average
- Can precompute a table and recycle it or simply jitter on the fly



Comparison

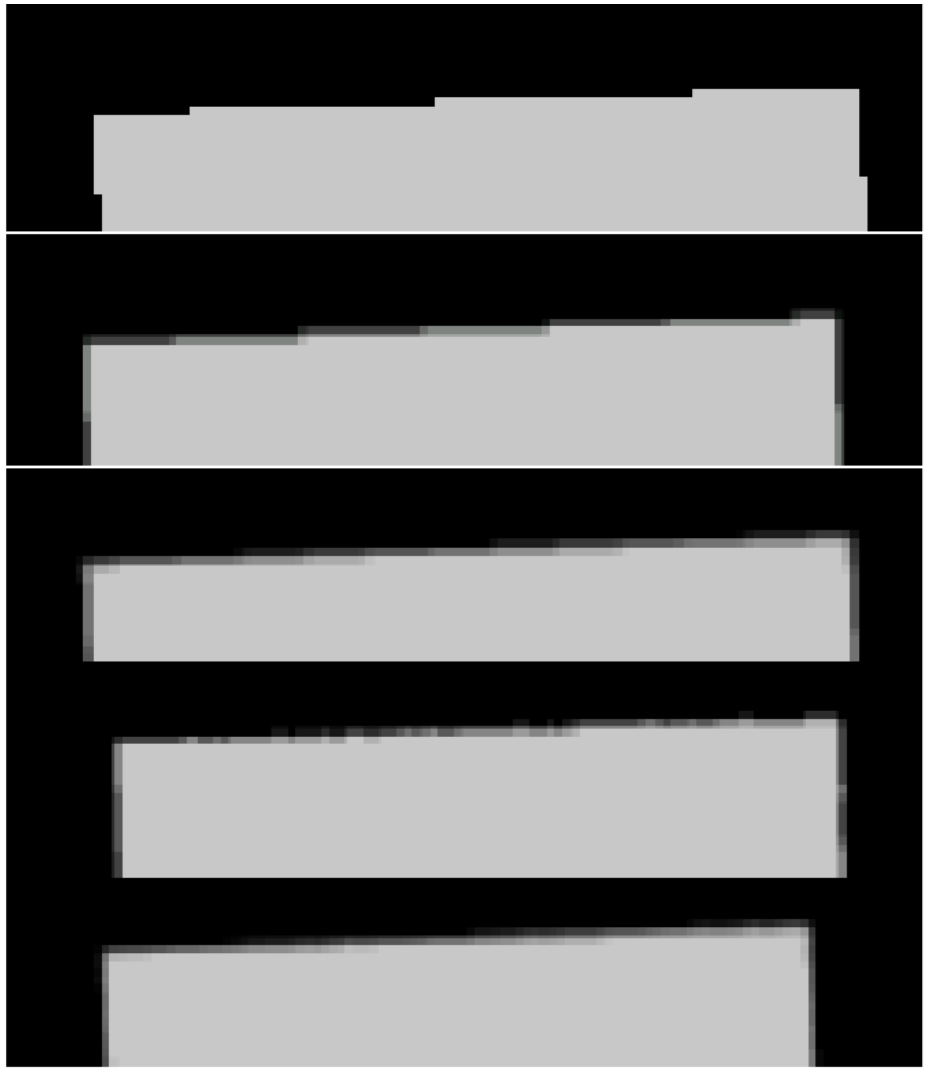
Regular, 1x1

Regular 3x3

Regular, 7x7

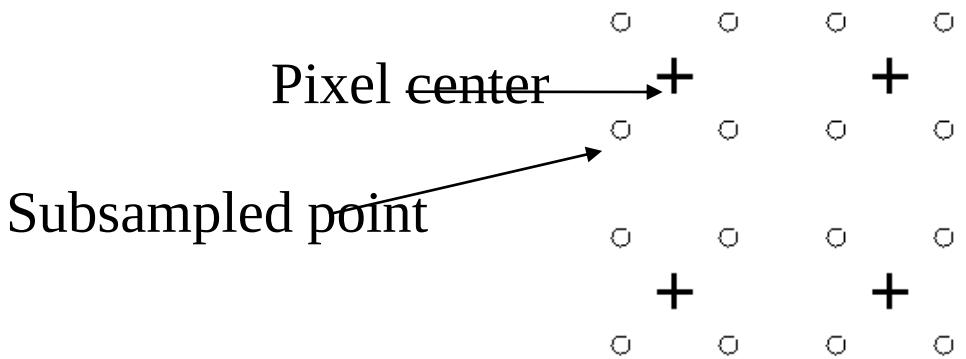
Jittered, 3x3

Jittered, 7x7



Accumulation Buffer (A-Buffer)

- Use a buffer that has the same resolution as the original image
- To obtain a 2x2 sampling of a scene, 4 images are made by shifting the frame buffer horizontally/vertically for half a pixel
- The results are accumulated and the final results are obtained by averaging
- We can recycle the vertex attributes

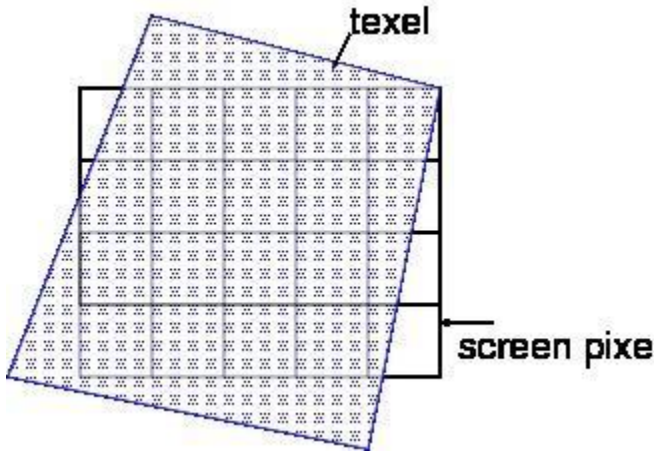


Aliasing of textures

- Happens when the camera is zoomed too much into the textured surface (magnification)
- Several texels (pixels of the texture) covering a pixel's cell (minification)

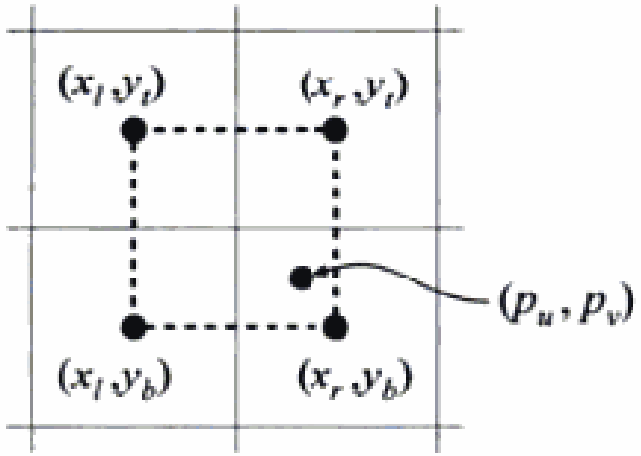
Texture Magnification

- Zooming too much into a surface with a texture
- One texel covering many pixels



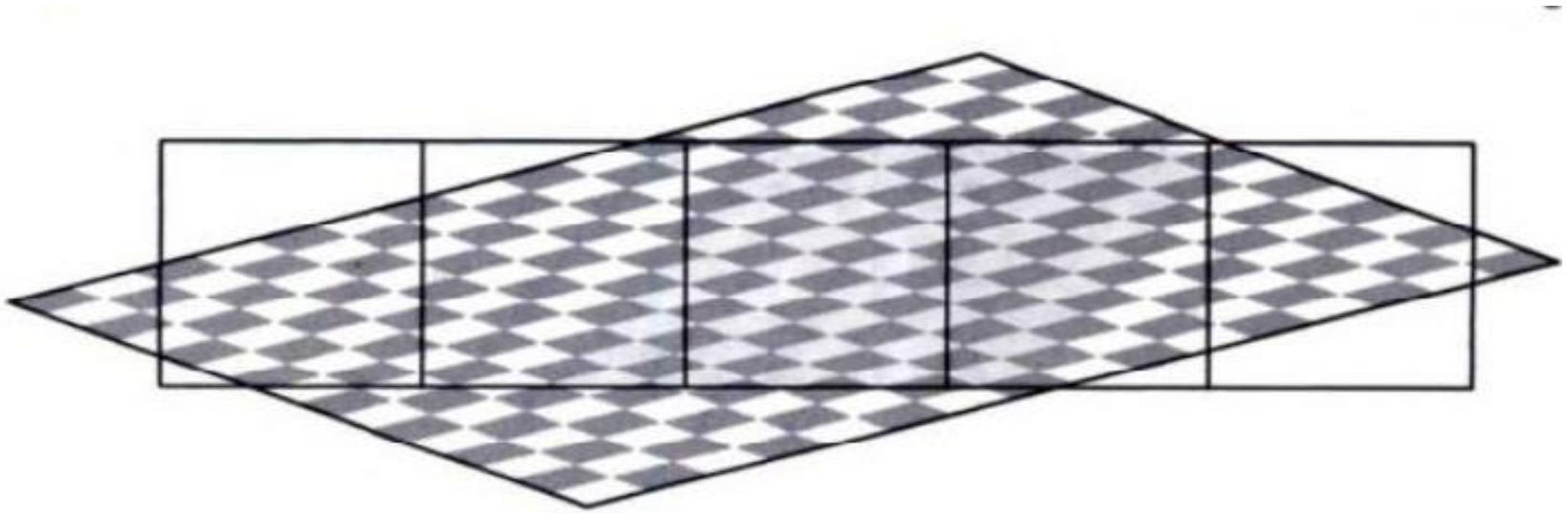
Bilinear Interpolation

- Mapping the pixel centre to the uv coordinates
- Computing the pixel colour by interpolating the surrounding texel values



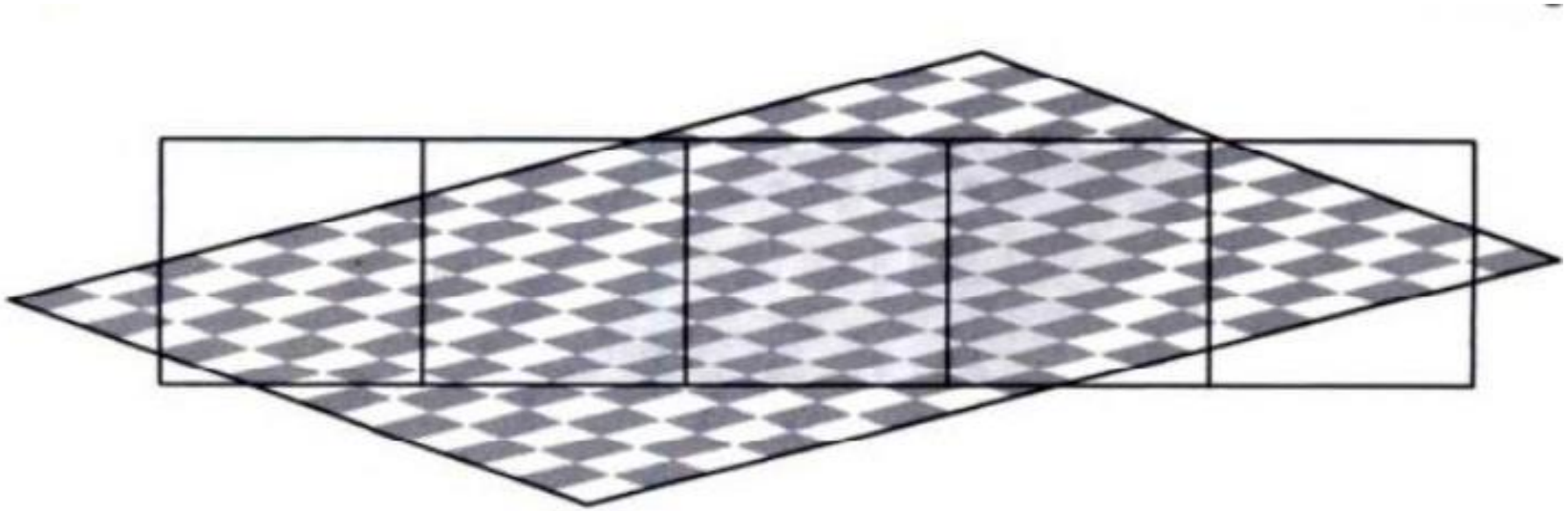
Texture Minification

- Many texels covering a pixel's cell
  - Results in aliasing (remember Nyquist limit)
  - The artifacts are even more noticeable when the surface moves



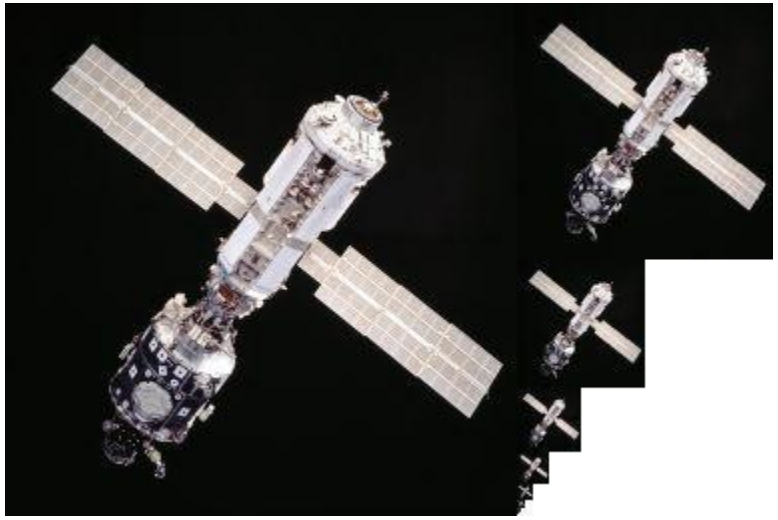
Texture Minification

- We can do subsampling as before
- But actually we know the texture pattern in advance
  - Mipmapping



MIP map

Multum In Parvo = Many things in a small place

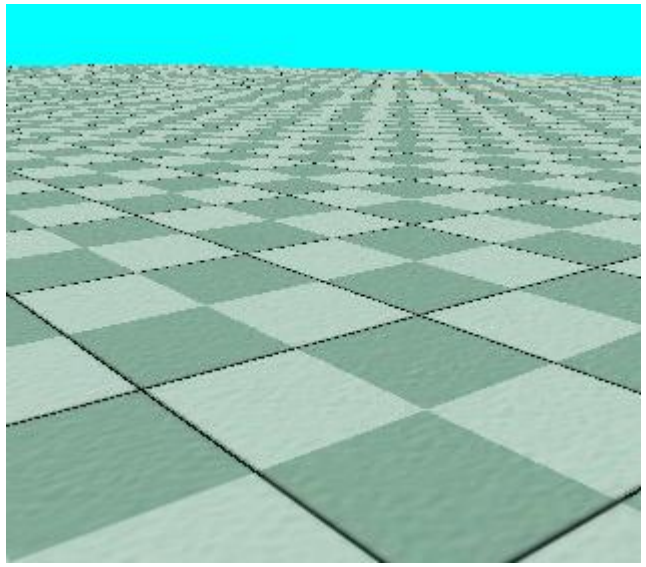


- Produce a texture of multiple resolutions
- Switch the resolution according to the number of texels in one pixel
- Select a level that the ratio of the texture and the pixel is 1:1

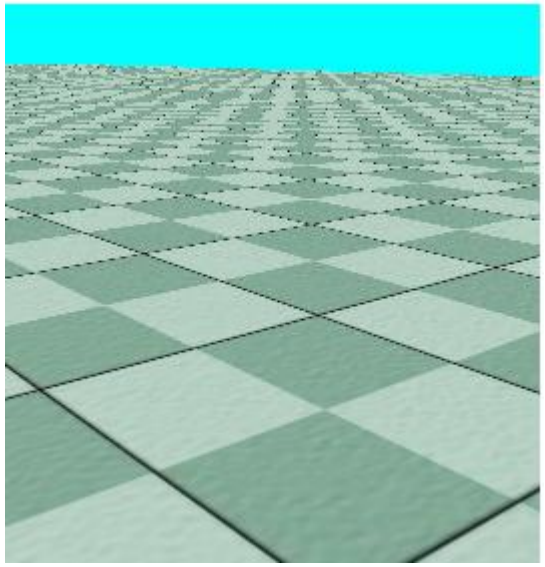
Texture Minification

Multiple textures in a single pixel

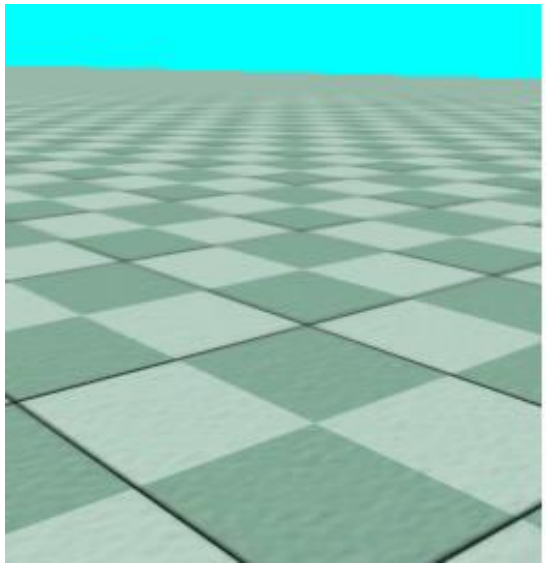
Solution:



Nearest neighbour



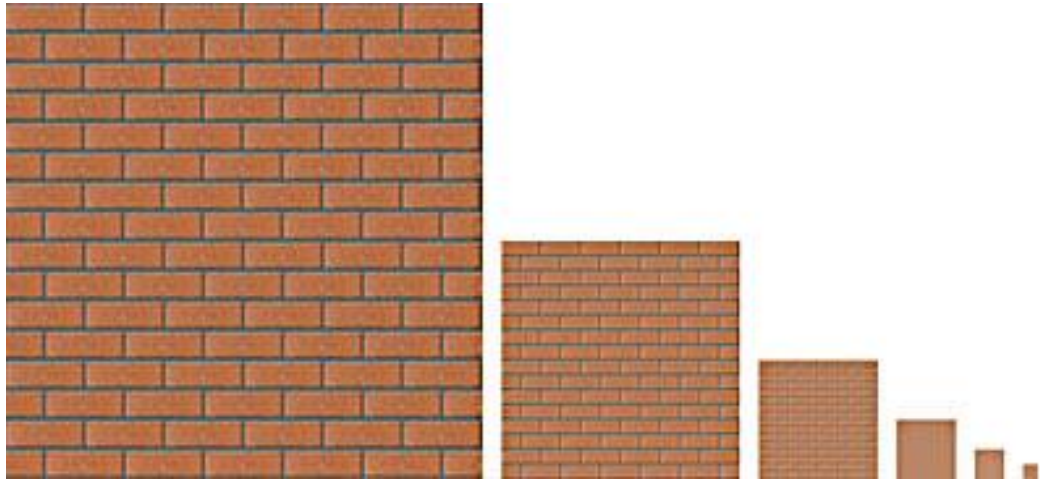
Bilinear blending



Mipmapping

Mipmapping

- Choosing a texture map resolution
  - Want high resolution textures for nearby objects
  - But high resolution textures are inefficient for distant objects
- Simple idea: Mipmapping
  - MIP: multum in parvo, “much in little”
  - Maintain multiple texture maps at different resolutions
  - Use lower resolution textures for objects further away (typically powers of 2)
    - How much extra memory does this require?
    - 1/4 + 1/16 + 1/64 … = 1/3
  - Example of “level of detail” (LoD) management common in CG
  - Reduces aliasing artifacts from down- and up-sampling



Mipmap for a brick texture

https://flylib.com/books/en/1.541.1.66/1/

Trilinear and Anisotropic Filtering

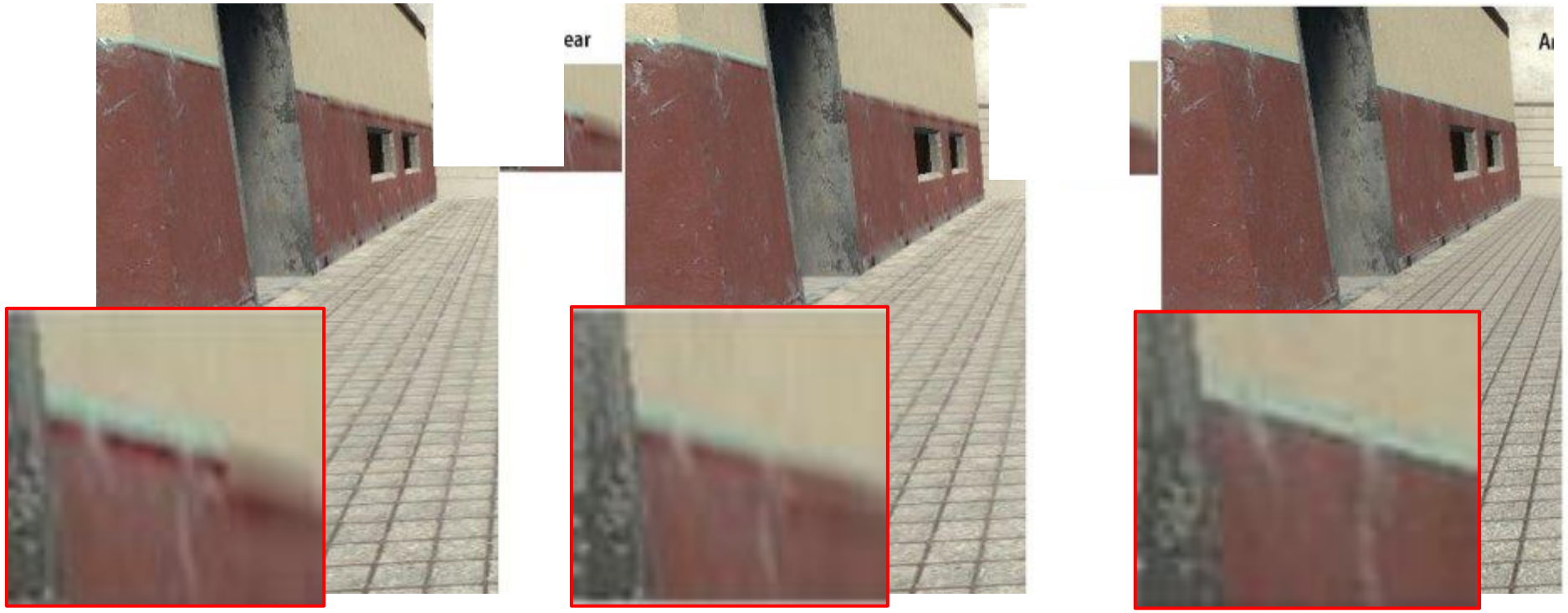
- But…
  - Transitions between levels in mipmaps can be jarring
  - Solution? Filter!
  - Trilinear filtering: Assume texture maps are at 2 ft, 4 ft, etc. Map a wall at, say, 3.5 ft by selecting the sequence of corresponding sample points for filtering in the two bracketing maps at 2 and 4 ft and blend them using scaling and linear interpolation
  - Doesn’t compensate for perspective distortion and the higher spatial frequencies it introduces

Trilinear and Anisotropic Filtering

Bilinear

Trilinear

Anisotropic



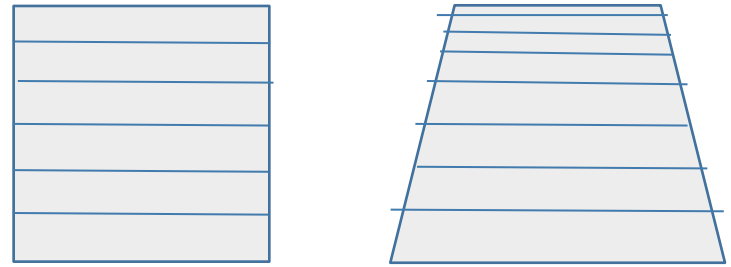
Filter Comparison

http://www.vgamuseum.info/index.php/news/item/869-anisotropic-filtering

Note the blue line accentuating the transition from red to beige receding in the distance

Trilinear and Anisotropic Filtering

- Mipmapping assumes orthogonal perspective to the texture (isotropic).
  - What happens to our rectangular texture as we rotate it from the xy plane (vertical wall) into xz (ground plane)?
- The rectangle becomes a trapezoid and spatial frequencies increase with z!



- Anisotropic filtering scales either the height or width of a mipmap by a ratio relative to the perspective distortion of the texture to undo the effect of the distortion
  - The ratio is dependent on the maximum sampling value specified, followed by taking the appropriate samples (in modern hardware, a max of 8 or 16 samples per pixel is common)
- The best part? Mipmapping and Anisotropic filtering can be enabled in ~1 line each!
- glGenerateMipmap(m_texture);
- glTexParameterf(m_texture, GL_TEXTURE_MAX_ANISOTROPY_EXT, numMaxSamples);
- Please use this simple code for final projects in TextureParameters.cpp!

Trilinear and Anisotropic Filtering (4/4)

https://youtu.be/WbIAtqMiJbo?t=39