Textures

Lecturer: Carol O'Sullivan

Credits: Some slides from Rachel McDonnell and Prof. Kavita Bala

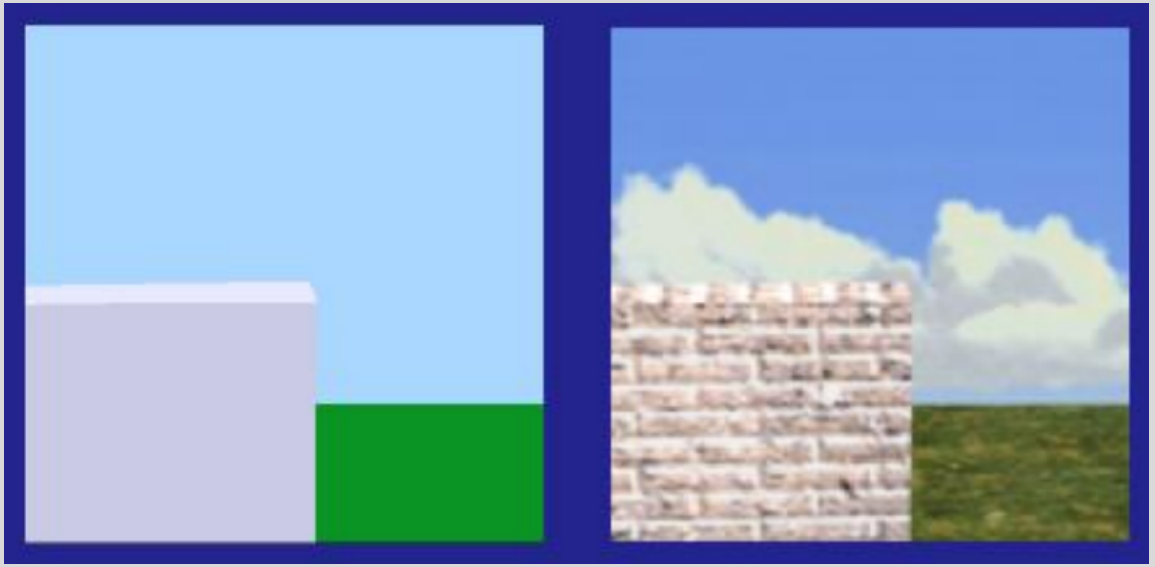
Introduction

- Adding lots of detail to our models to realistically depict skin, grass, bark, stone, etc., would increase rendering times dramatically, even for hardware-supported projective methods

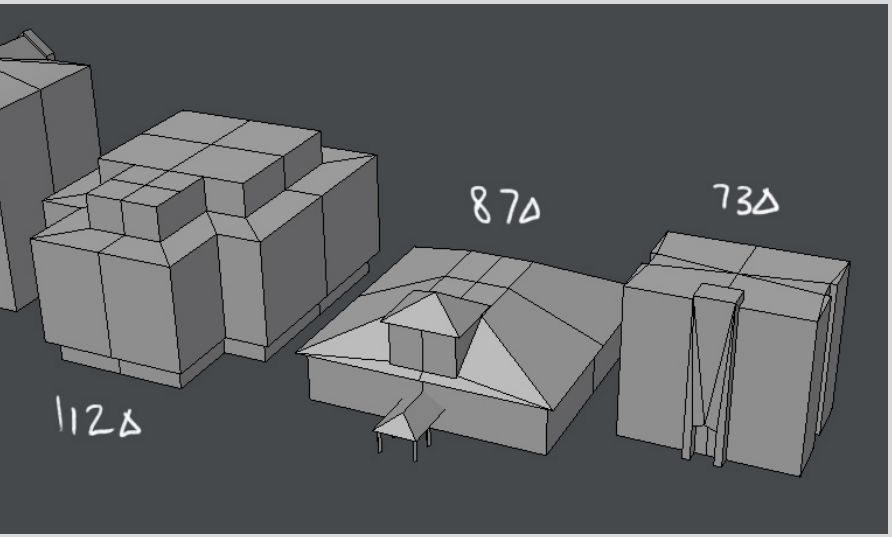
Unreal Engine 4

Texture Mapping

- Texture mapping allows you to take a simple polygon and give it the appearance of something much more complex
  - Due to Ed Catmull, PhD thesis, 1974
- Instead of calculating colour, shade, light, etc. for each pixel, we just paste images to our objects in order to create the illusion of realism

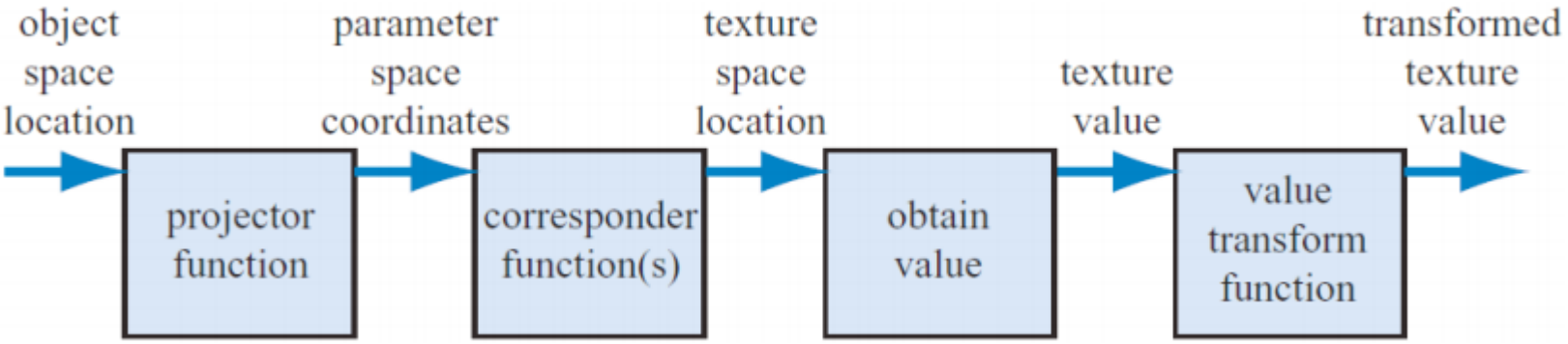


Example: Architectural Details

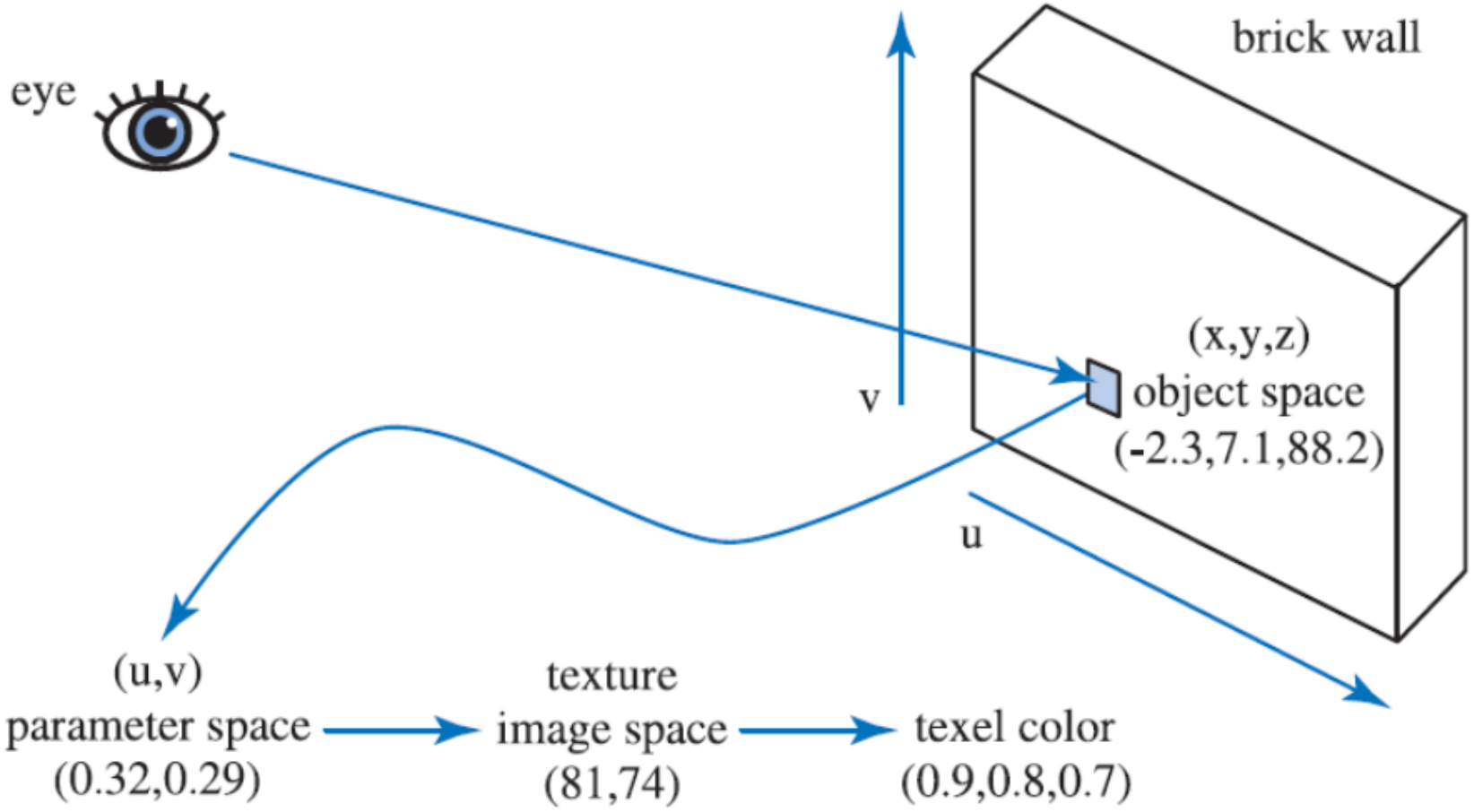


How does it work?

- Texturing works by modifying the values used in the shading equation.
- The pixels in the image texture are often called *texels,* to differentiate them from the pixels on the screen

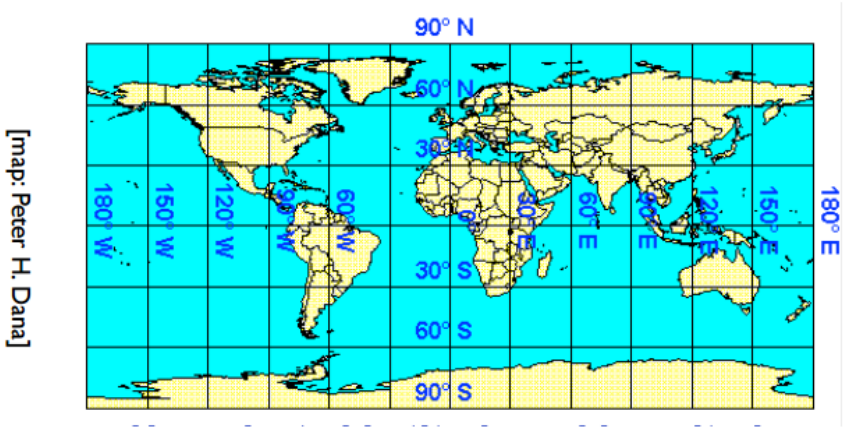


How does it work?



Projector functions

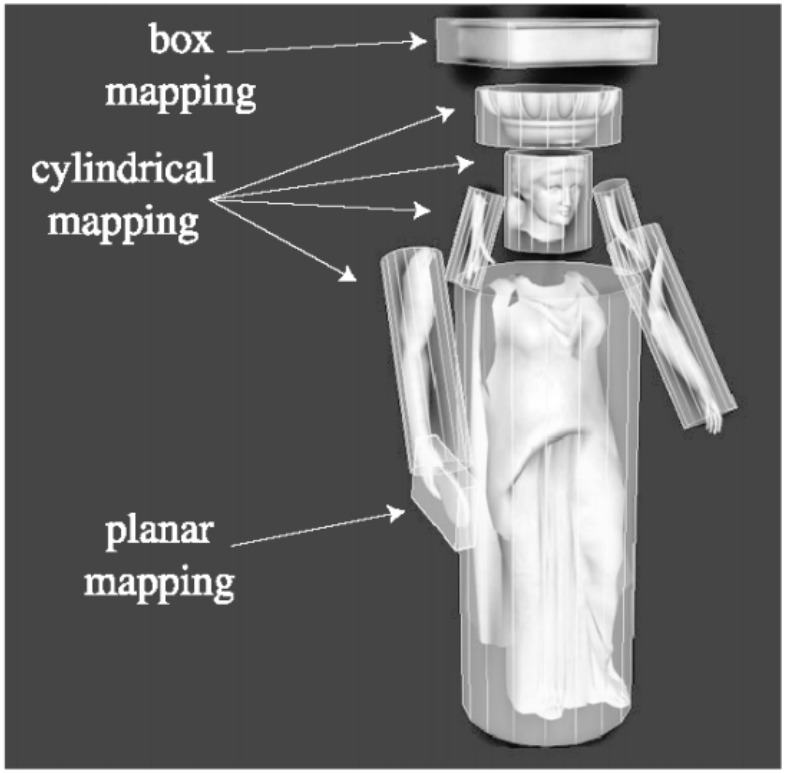
- Converting a 3D point in space into texture coordinates
- For a sphere: latitude-longitude coordinates –
  - φ maps point to its latitude and longitude



- *Usually done offline and stored with the vertex.*
- *But can be applied in the vertex shader, giving different effects such as animation (e.g., fire, water, blending between marble and skin textures to make a statue come to life*

Arbitrary Surfaces

- Non-parametric surfaces: project to parametric surface



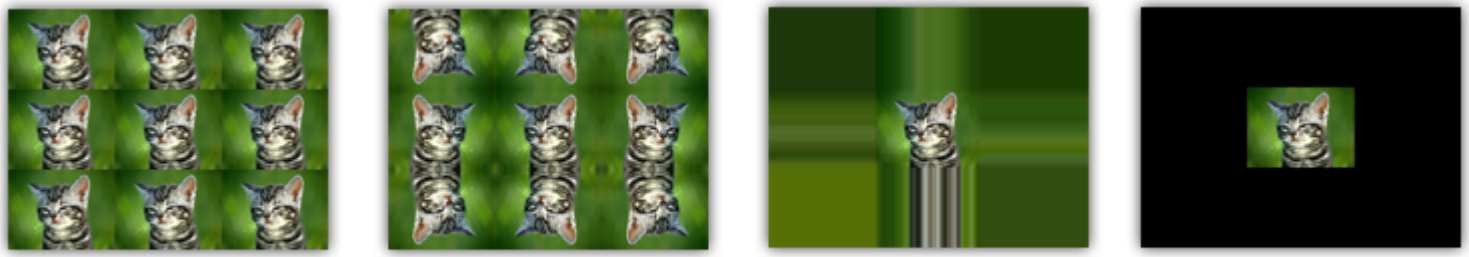
Artist intervention

Artist intervention

- An artist often manually decomposes the model into near-planar pieces
- Tools to help minimize distortion by unwrapping the mesh
- Goal: have each polygon be given a fairer share of a texture's area, while also maintaining as much mesh connectivity as possible

Corresponder Function

- Functions that convert parameter-space values to texture-space locations
- Select a subset of the image for texturing
- Decide what happens at boundaries
- In OpenGL: wrapping mode

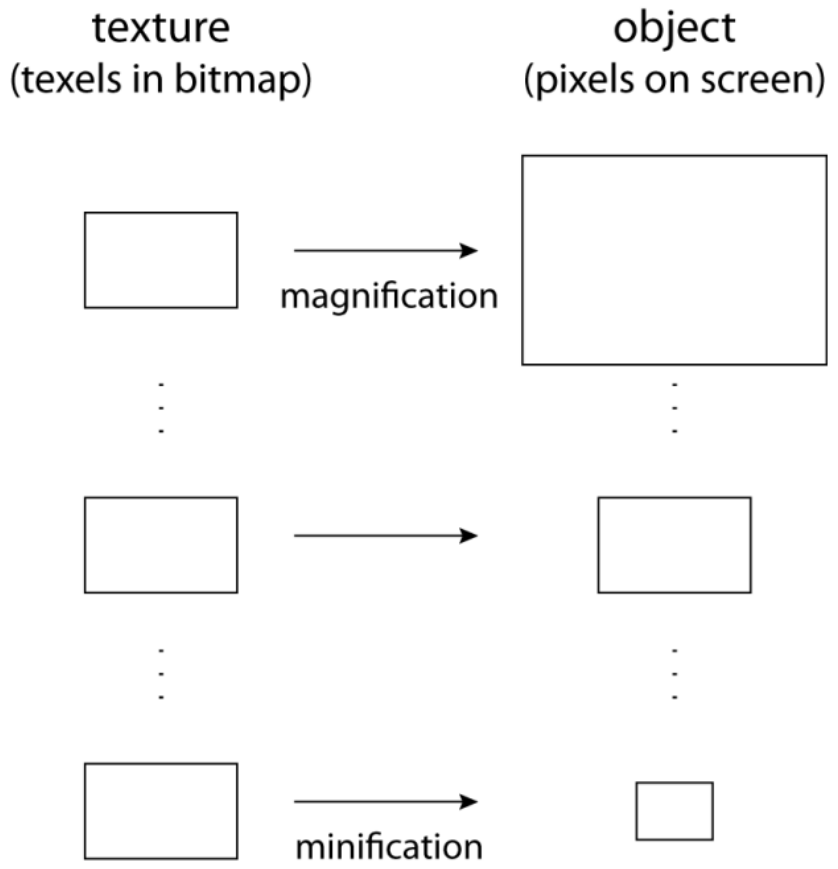


- *Wrap: Repeats*
- *Mirror – Repeats but mirrored every other time; continuity across edges*
- *Clamp: Clamped to edge of texture*
- *Border: Clamped to border colour*

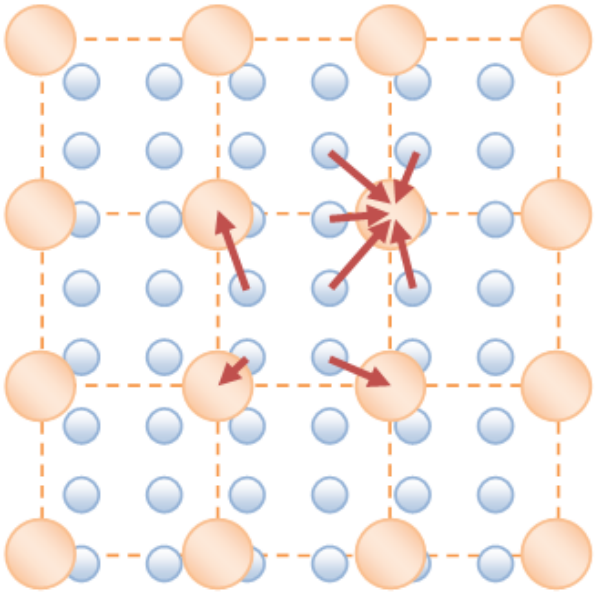
Magnification & Minification

- If viewer is close: Object gets larger → Magnify texture
- "Perfect" distance: Not always "perfect" match (misalignment, etc.)
- If viewer is further away: Object gets smaller → Minify texture
- Problem with minification?
  - efficiency (esp. when whole texture is mapped onto one pixel!)

What if the projected texture square does not match the screen resolution?



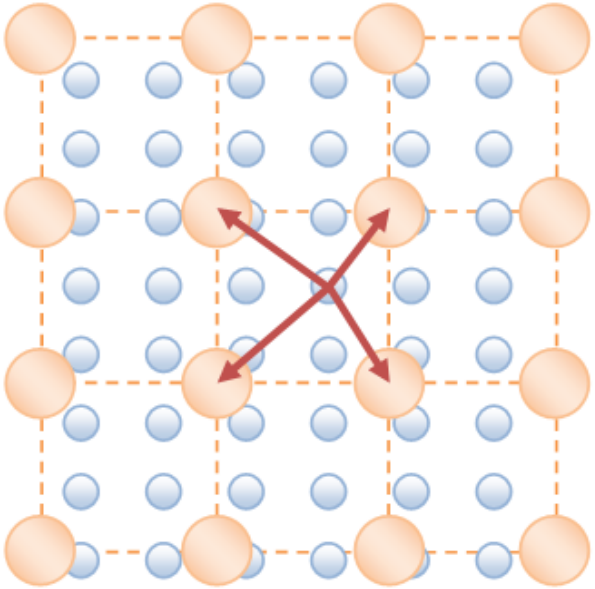
Magnification



texel

fragment

Magnification – Nearest Point Sampling



texel

fragment

Magnification – Bilinear Interpolation

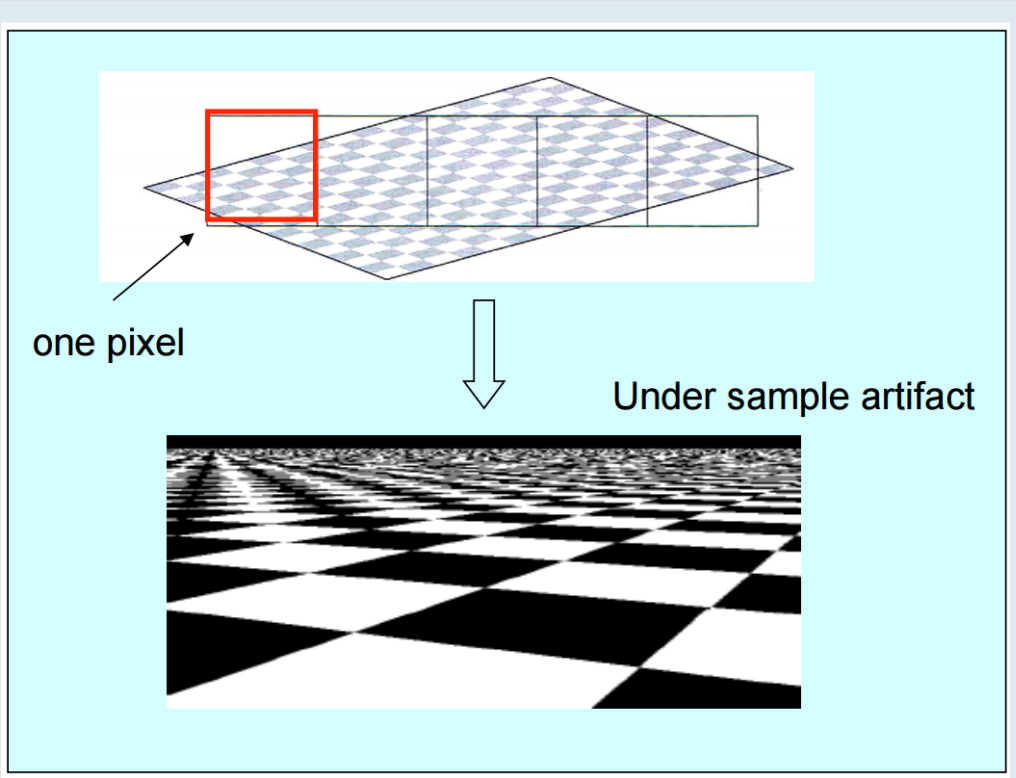
https://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

Magnification

Magnification

Magnification aliasing – walls are lower resolution than on-screen pixels
(Tomb Raider, Eidos Interactive, 1996)

Minification



- Several texels cover one pixel (under sampling happens)
- Solution: Either increase sampling rate or reduce the texture Frequency

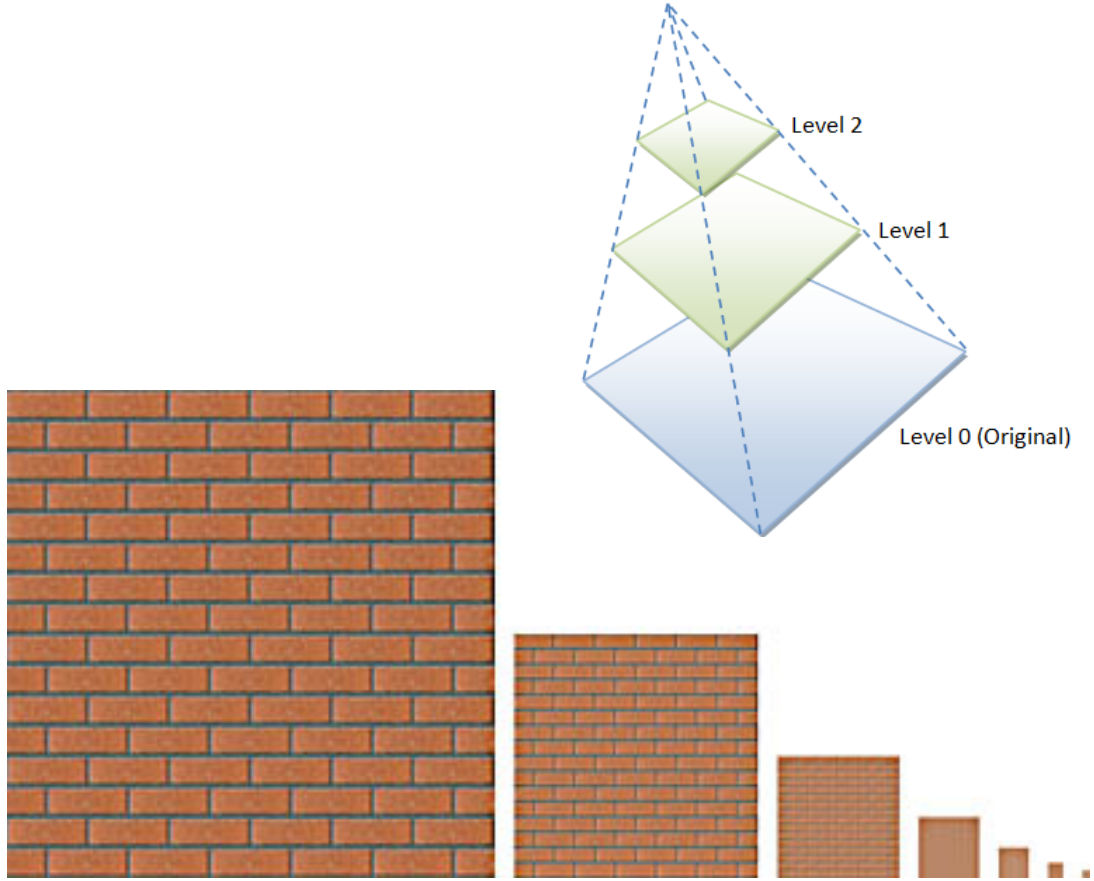
Minification

Visible flicker when camera moves

Minification aliasing – trees are higher resolution than on-screen pixels (*Combat Mission, Battlefront.com, 1999)*

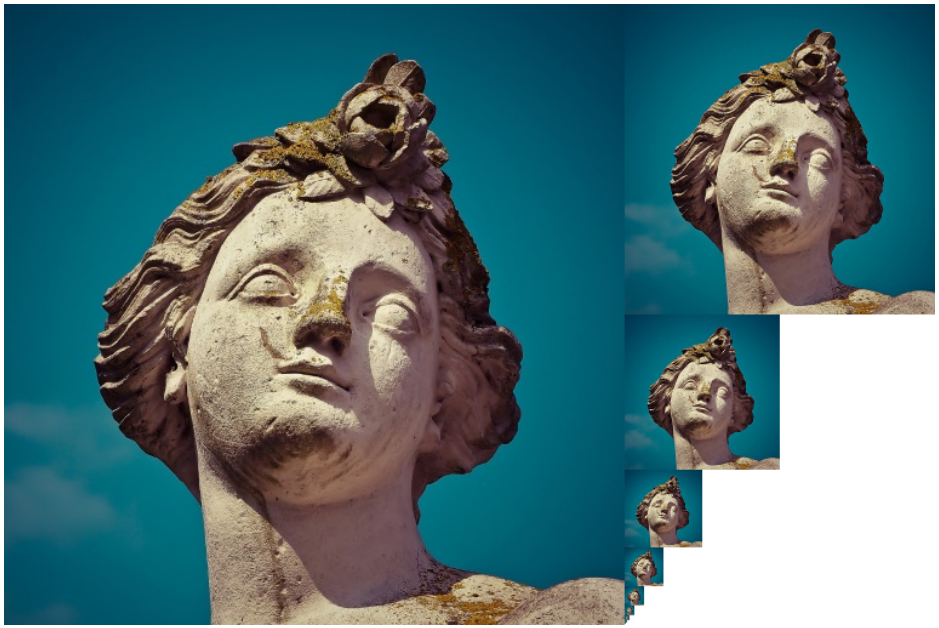
Solution: MIP maps

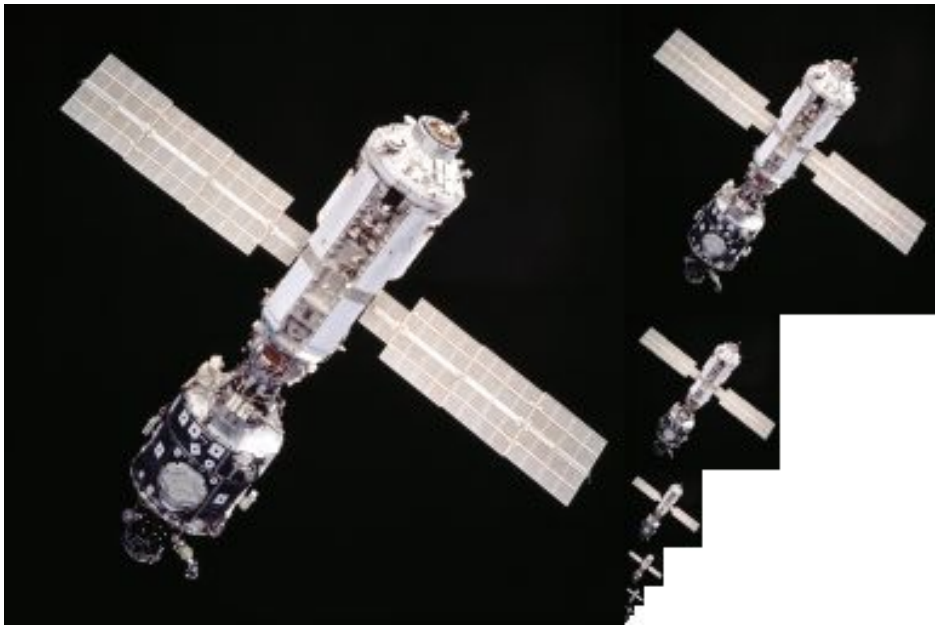
- Pre-calculated, optimized collections of images based on the original texture
  - Dynamically chosen based on depth of object (relative to viewer)
  - Supported by todays hardware and APIs



MIP – *multum in parvo*
"many things in a small place"

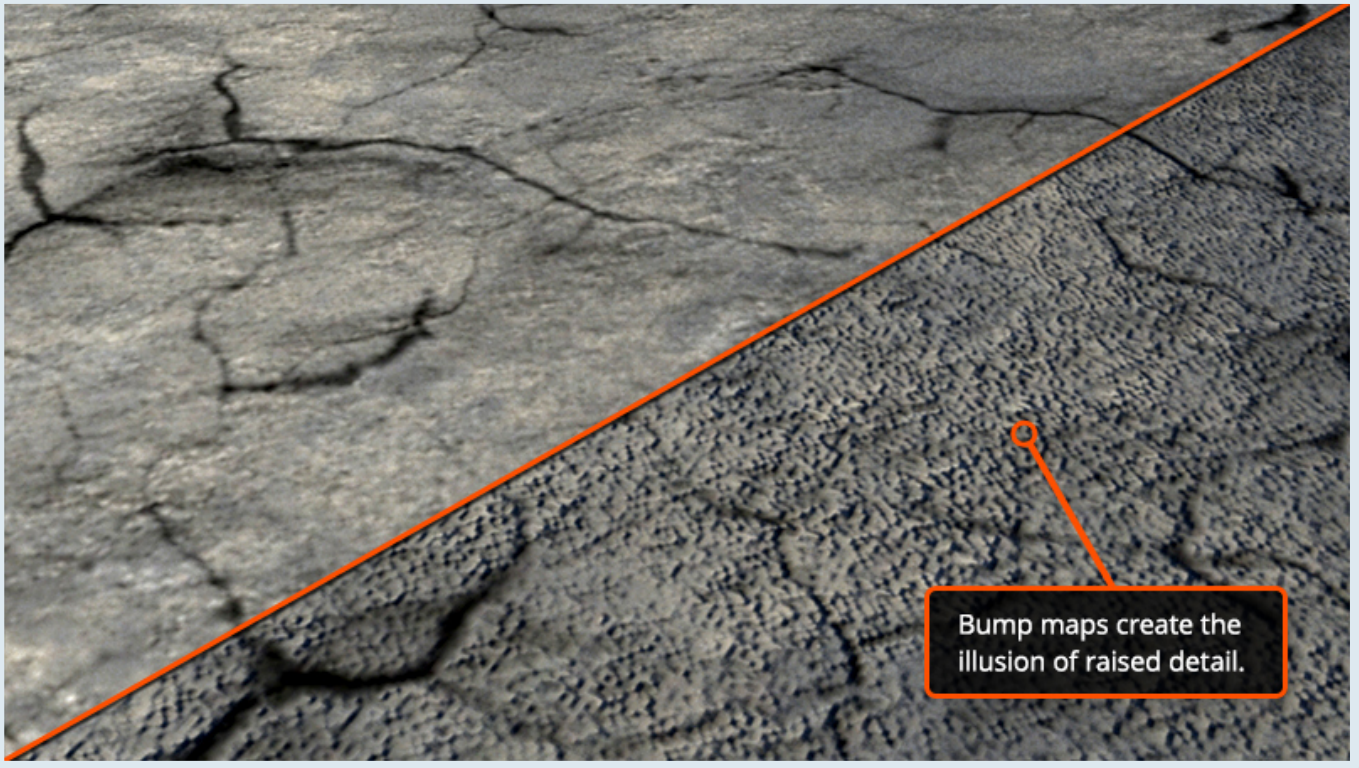
Mip map chains





Bump Maps

- One of the reasons why we apply texture mapping: Real surfaces are not flat but often rough and bumpy.



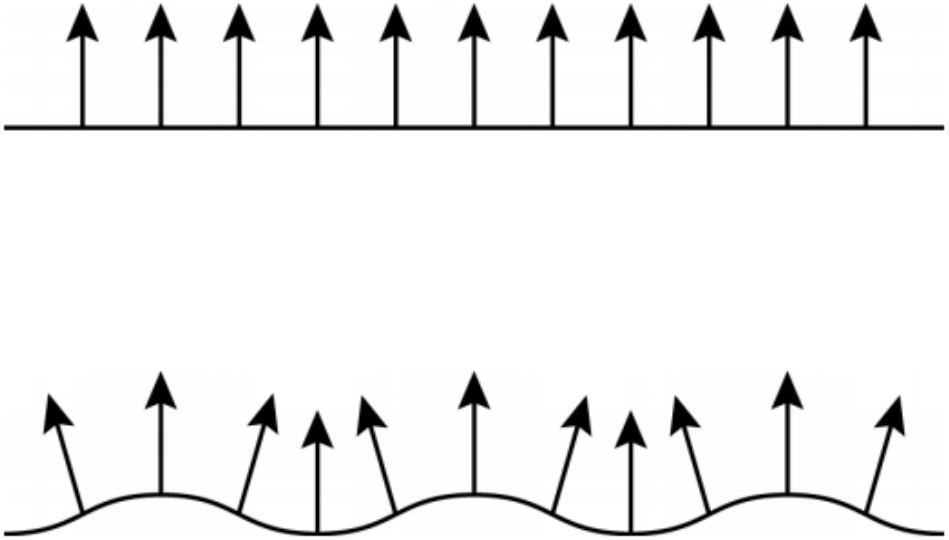
http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Bump Maps

- Bump maps create the illusion of depth on the surface of a model using a very simple lighting trick.
- No additional resolution is added to the model as a result of a bump map.
- Typically, bump maps are grayscale images that are limited to 8-bits of color information.
  - That's only 256 different colors of black, gray or white.
  - These values in a bump map are used to tell the 3D software basically two things: Up or down.

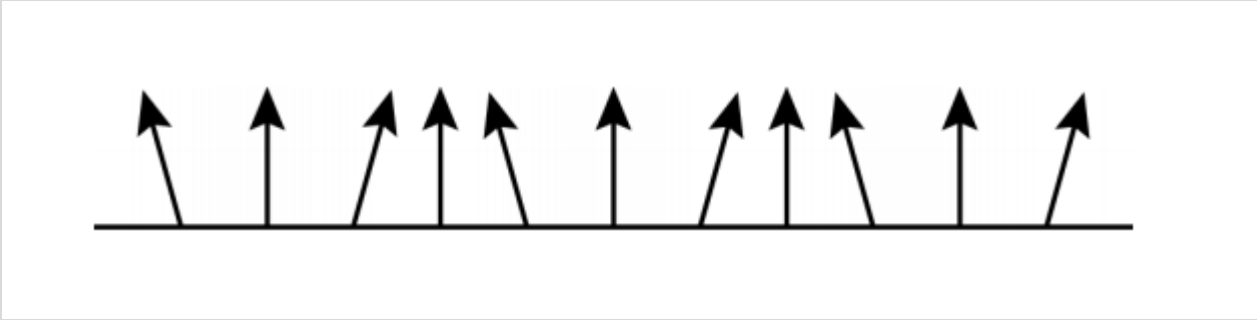
Normal mapping

- Normal maps can be thought of as a new, improved type of bump map
  - also fakes details
- These bumps cause (slightly) different reflections of the light



Normal mapping

- Instead of mapping an image or noise onto an object, we can also apply a normal map
  - this is a 2D or 3D array of vectors
  - these vectors are added to the normals at the points for which we do shading calculations

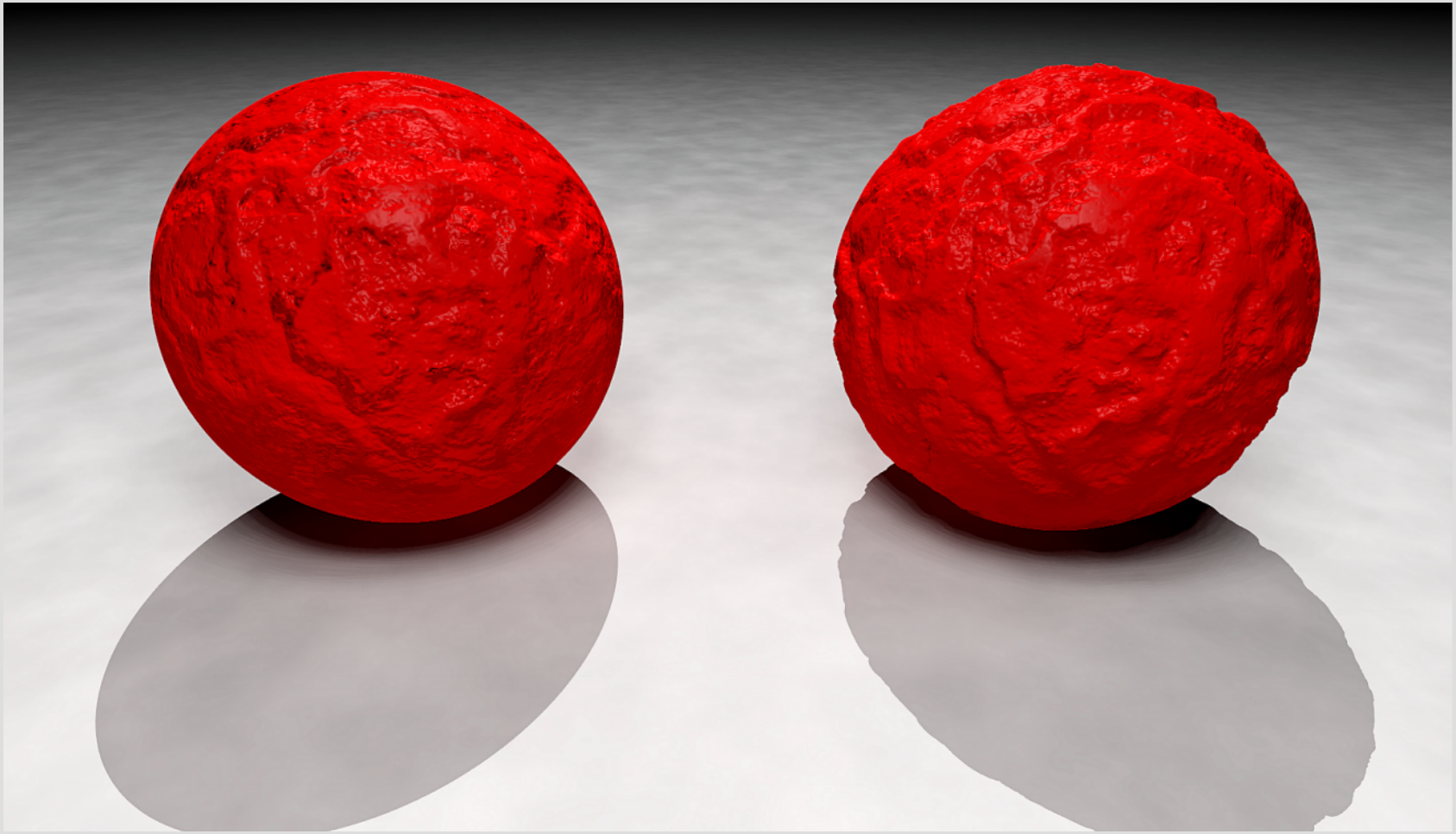


- The effect of normal mapping is an apparent change of the geometry of the object.

Normal mapping

- Instead of using a texture to change colour component in the illumination equation, we access a texture to modify the surface normal.
- Geometric normal remains the same; we merely modify the normal used in the lighting equation
- Implemented by modifying the per-pixel shading routine
- A normal map uses RGB information that corresponds directly with the X, Y and Z axis in 3D space

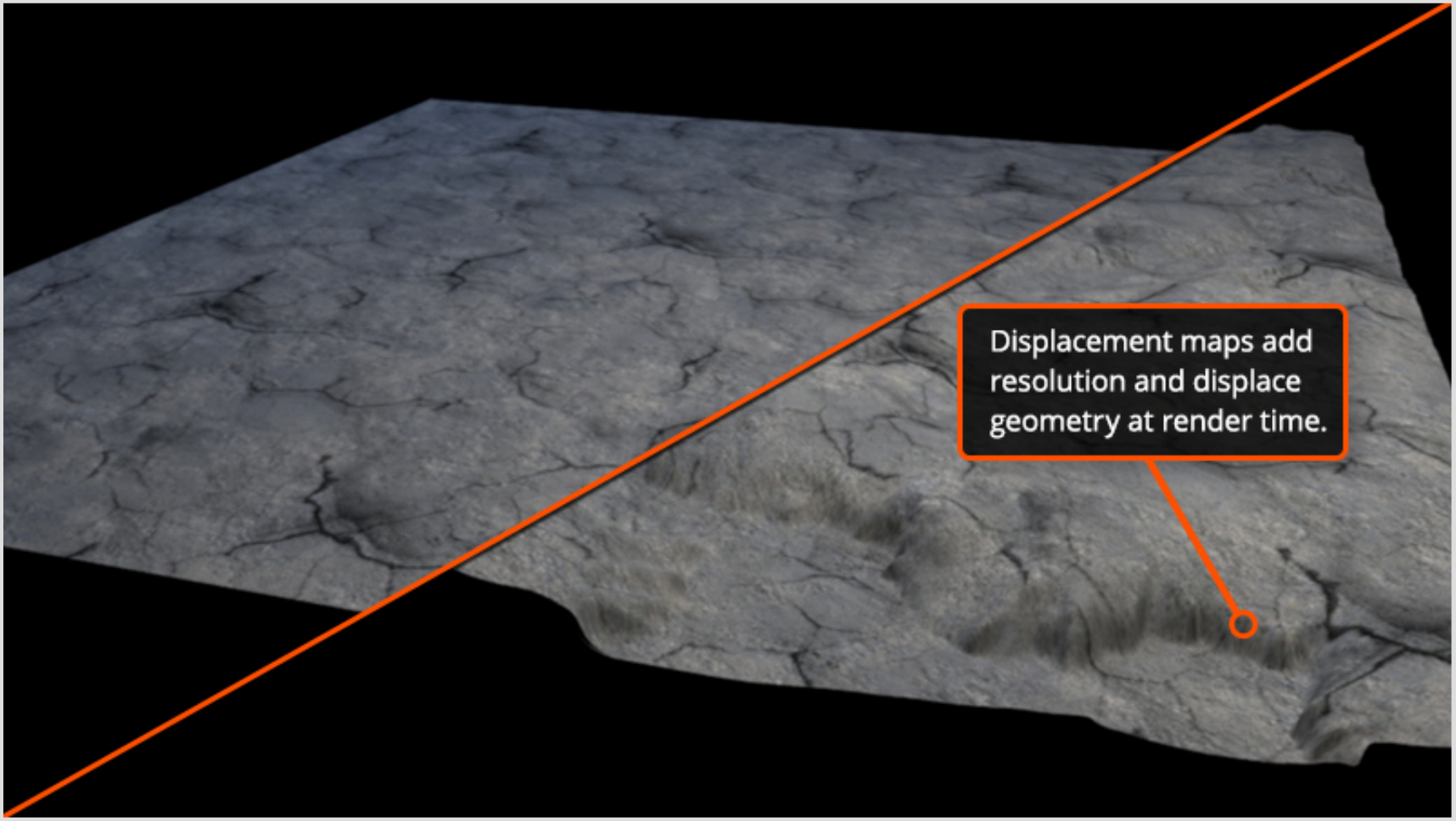
What's the main problem?



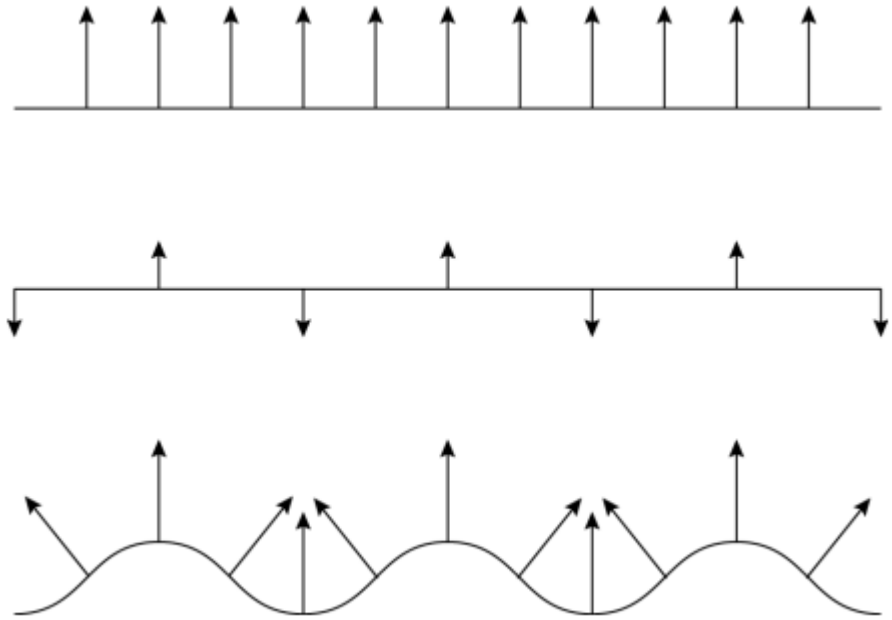
Displacement mapping

- To overcome this shortcoming, we can use a displacement map.
  - this is also a 2D or 3D array of vectors, but here the points to be shaded are actually **displaced**.
- Normally, objects are refined using the displacement map, giving an increase in storage requirements

Displacement Maps



Displacement Mapping



Problem?

- More challenging:
  - Collision detection
  - Object intersection
  - Foot placement

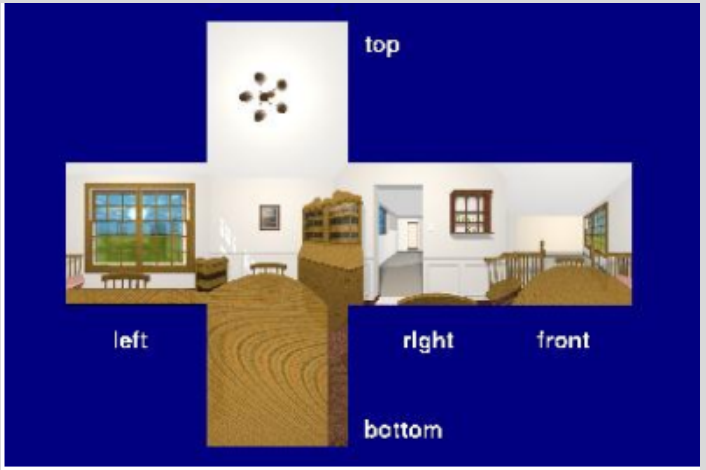
Environment Mapping

- When you look at a highly reflective object such as a chrome sphere, what you see is not the object itself but how the object reflects its environment
- ... why not use this to make objects appear to reflect their surroundings specularly?

http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter07.html

Environment Mapping

- Idea:
  - place a cube around the object
  - project the environment of the object onto the planes of the cube in a pre-processing stage
  - this is our texture map
- During rendering:
  - compute a reflection vector,
  - then use that to look-up texture values from the cubic texture map.



Basic Steps

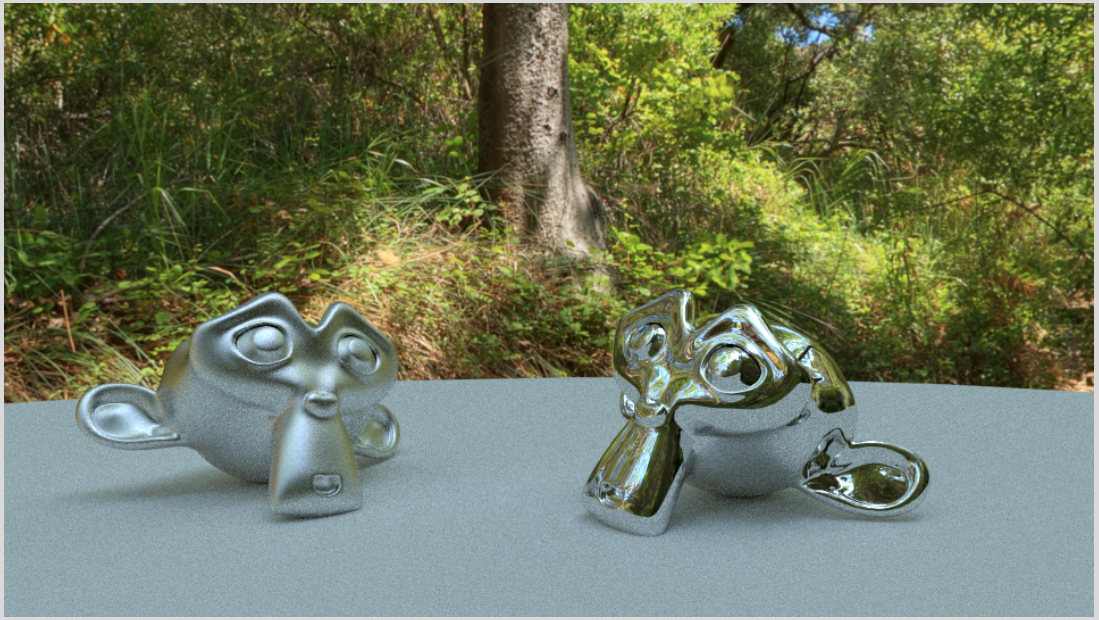
- Create a 2D environment map
- For each pixel on a reflective object, compute the normal
- Compute the reflection vector based on the eye position and surface normal
- Use the reflection vector to compute an index into the environment texture
- Use the corresponding texel to color the pixel

Environment Mapping

- Ideally, every environment-mapped object in a scene should have its own environment map
  - In practice, objects can often share environment maps with no one noticing
- In theory, you should regenerate an environment map when objects in the environment move, or when the reflective object using the environment map moves significantly relative to the environment
  - In practice, convincing reflections are possible with static environment maps.

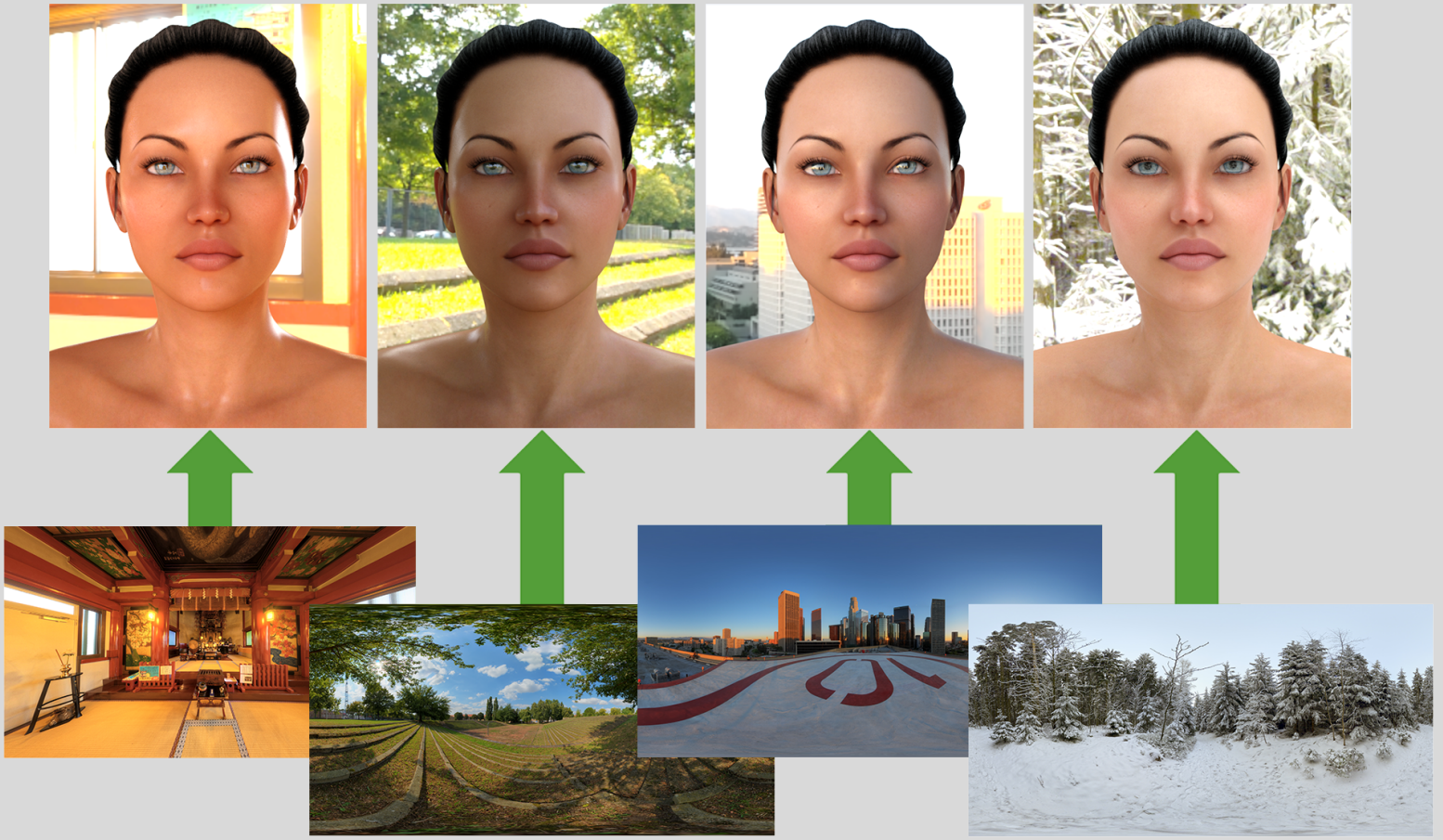
Environment Mapping

- Because environment mapping depends solely on direction and not on position, it works poorly on flat reflective surfaces such as mirrors, where the reflections depend heavily on position.
- In contrast, environment mapping works **best on curved surfaces**.



examples: http://www.humus.name/index.php?page=Textures

Environment Mapping



http://docs.daz3d.com/doku.php/public/software/dazstudio/4/new_features/start

Control Maps

- The assumption is that each object has the same reflectivity over its entire surface.
  - But this doesn't necessarily have to be the case!
- You can create more interesting effects by encoding reflectivity in a texture.
  - This approach allows you to vary the reflectivity at each fragment, which makes it easy to create objects with both specular and non-specular parts.
- Control maps are especially important because they leverage the GPU's efficient texture manipulation capabilities.
- Control maps give artists more control over effects without requiring a deep understanding of the implementation

Procedural textures

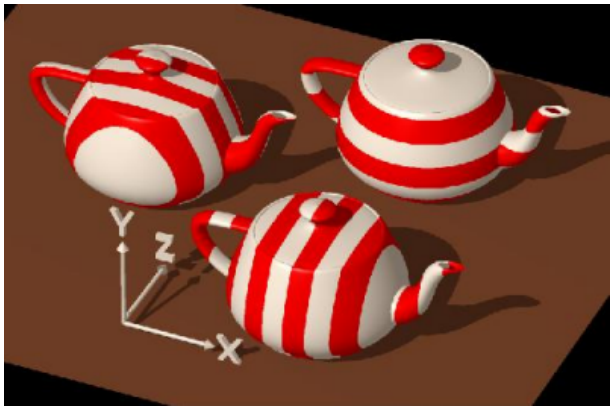
- We can also use a mathematical procedure to create a 3D texture

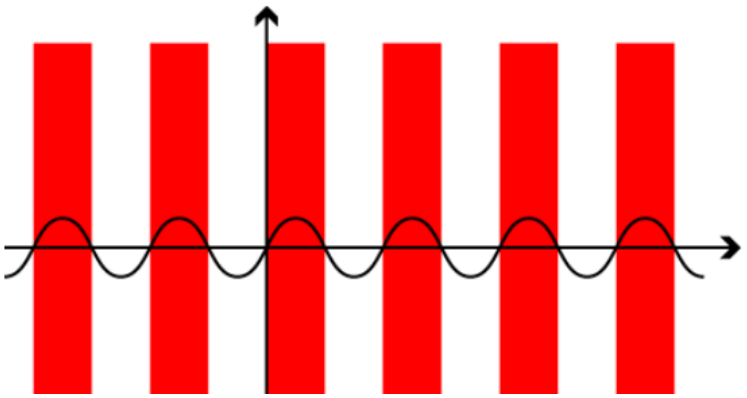
- Then we use the coordinates of each point in our 3D model to calculate the appropriate colour value using that procedure

Procedural textures

- A simple example: stripes along the X-axis

```
stripe( x_p, y_p, z_p )
{
  if ( sin x_p > 0 )
    return color0;
  else
    return color1;
  }
}
```





Extra Reading

- Environment Mapping: http://www.pauldebevec.com/Probes/
- Real-time Rendering, 3rd Edition, Akenine-Moller

OpenGL Implementation

- Preparing the “texture” involves the following:
  - Load image file with image loader library
  - Decoded image RGB[A] bytes are in main memory
  - Create OpenGL texture object
  - Copy image bytes into texture (-> graphics memory)
  - Set up texture filtering properties of texture
  - Set up texture wrapping properties of texture
  - Bind texture into active texture slot number 0

Image-Loader Library

- Sean Barrett's stb_image.h is simple and small and doesn't need linking
  - https://github.com/nothings/stb
- **libPNG** is kind of complicated to use, but okay
- **SOIL** loads and creates GL textures too
- You can write your own – RAW formats and TGA are pretty simple to load manually
- It will help a lot to know how to read/write raw binary image files

GL Functions to Load Texture

- `glGenTextures`
- `glActiveTexture (GL_TEXTURE0);`
- `glBindTexture`
- `glTexImage2D`
  - `GL_RGBA or GL_RGB`
- `glTexParameteri`
  - `GL_TEXTURE_WRAP_S`
  - `GL_TEXTURE_WRAP_T`
  - `GL_TEXTURE_MAG_FILTER`
  - `GL_TEXTURE_MIN_FILTER`

Always call before binding a texture

Copies actual image data into texture

Must call 4 times and set these 4 parameters or may not work at all

https://open.gl/textures