5. Textures & Mipmaps

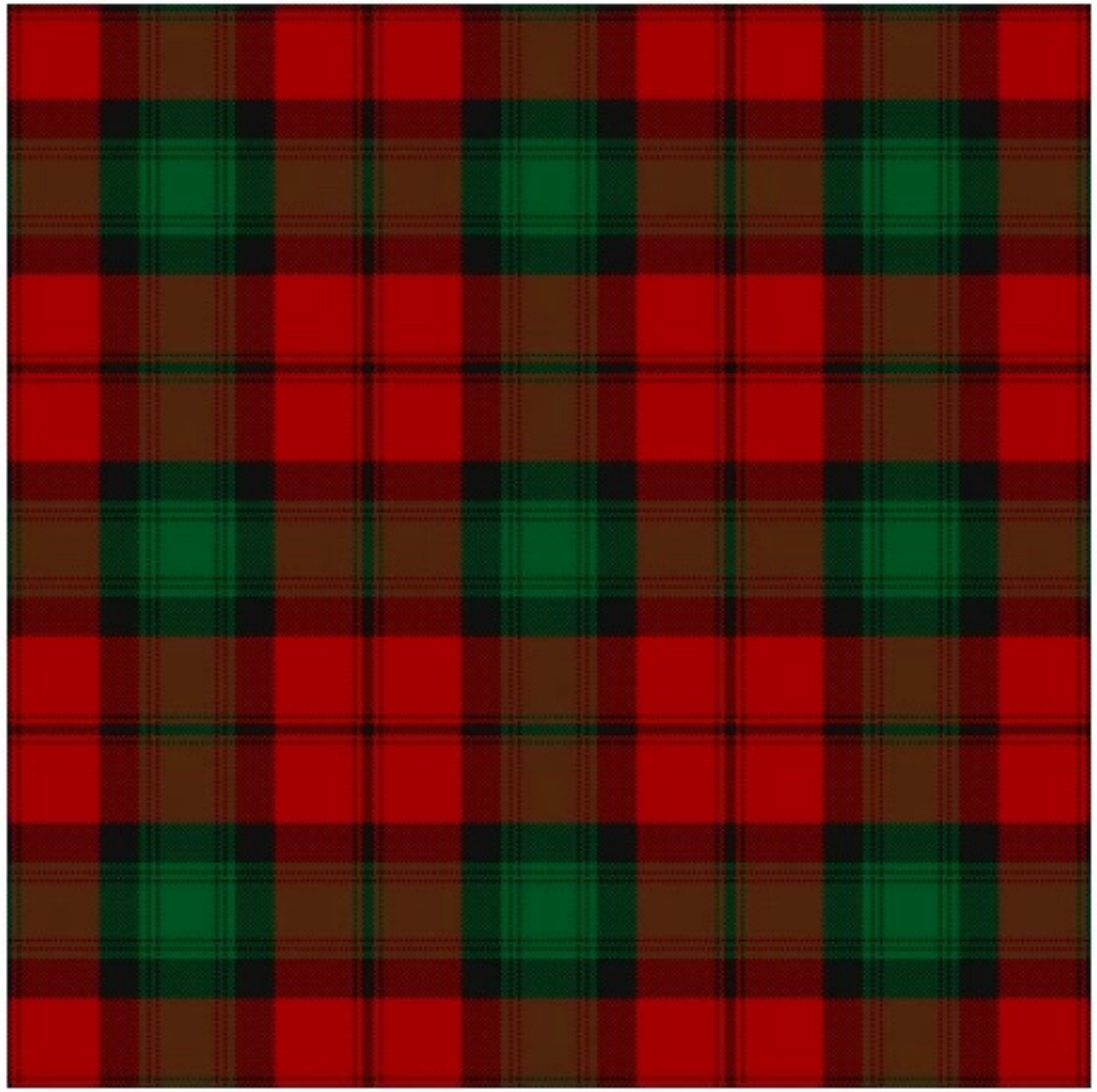
Dr. Hamish Carr & Dr. Rafael Kuffner dos Anjos

Agenda

- What are textures?
- How to sample from textures?
- Texture interpolation
- Practical uses of textures

Textures & Mapping

- An image mapped to a surface
- Substitutes for geometric detail
- $f: M \rightarrow \mathbb{R}^n$
  - M is the manifold
- $\mathbb{R}^n$ is the data



So . . .

Any property we wish to store on a surface

We convert what we want to store to an image

Then use texture coordinates to access

Adding Textures

Given texture coords for the pixel we can interpolate texels.

Start with a point p

Convert to surface parameters (s,t)	Convert to texel indices (i,j)	Retrieve texel colour

Analytic Solution: Spherical coordinates

Given x,y,z on a sphere

Find (θ,φ) (lat., long.)

Then (i,j) texture indices

$$x = r\cos\phi\cos\theta$$

$$y = r\cos\phi\sin\theta$$

$$\frac{y}{x} = \frac{r\cos\phi\sin\theta}{r\cos\phi\cos\theta} = \frac{\sin\theta}{\cos\theta} = \tan\theta$$

so:

$$\theta = \arctan\frac{y}{x}$$ (Use C function atan2(y,x))

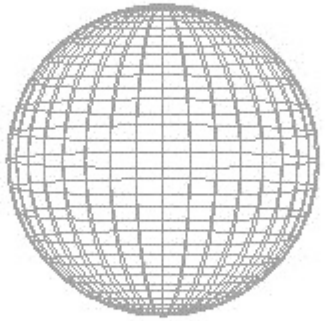
And:

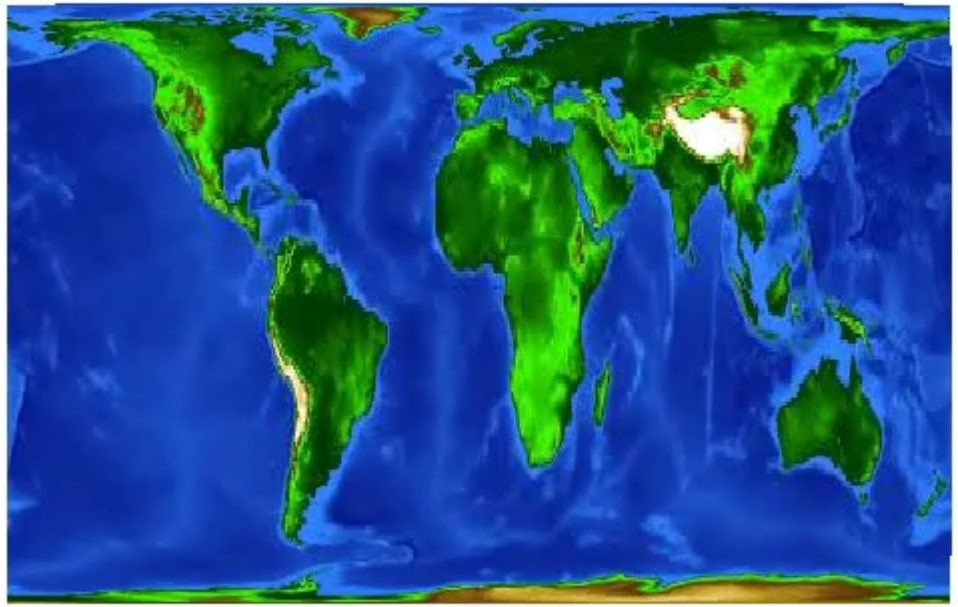
$$z = r\sin\phi$$

so

$$\phi = \arcsin\frac{z}{r}$$

Textured Sphere with cylindrical projection





Texture Coordinates

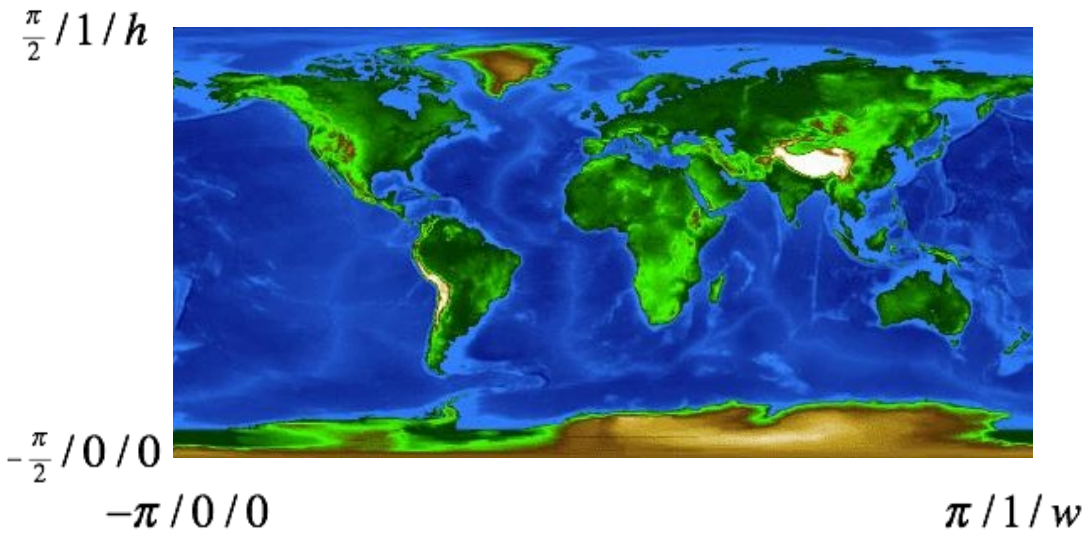
Assume that texture has height h, width w

Point	(ϕ,θ)	(s,t)	(i,j)
Top Left	$\left(\frac{\pi}{2},-\pi\right)$	$(1,0)$	$(h,0)$
Top Right	$\left(\frac{\pi}{2},\pi\right)$	$(1,1)$	(h,w)
Bottom Left	$\left(-\frac{\pi}{2},-\pi\right)$	$(0,0)$	$(0,0)$
Bottom Right	$\left(-\frac{\pi}{2},\pi\right)$	$(0,1)$	$(0,w)$

We can compute (i,j) as follows:

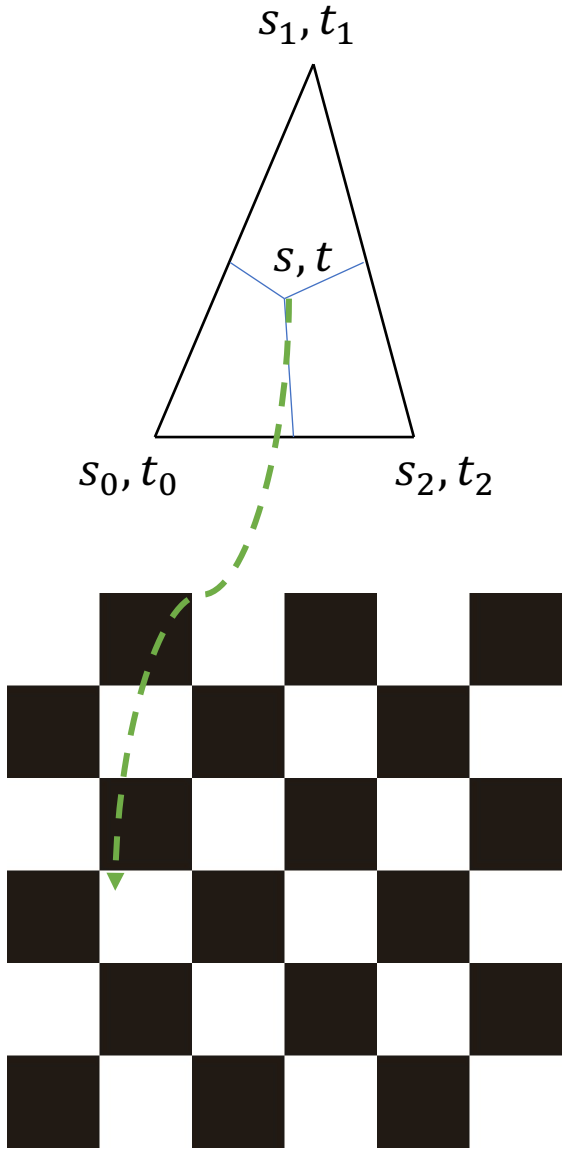
$$s=\frac{(\theta+\pi)}{2\pi}=\frac{\theta}{2\pi}+\frac{1}{2} \qquad i=h\cdot s=h\left(\frac{\theta}{2\pi}+\frac{1}{2}\right)$$

$$t=\frac{\left(\phi+\frac{\pi}{2}\right)}{\pi}=\frac{\phi}{\pi}+\frac{1}{2} \qquad j=w\cdot t=w\left(\frac{\phi}{\pi}+\frac{1}{2}\right)$$

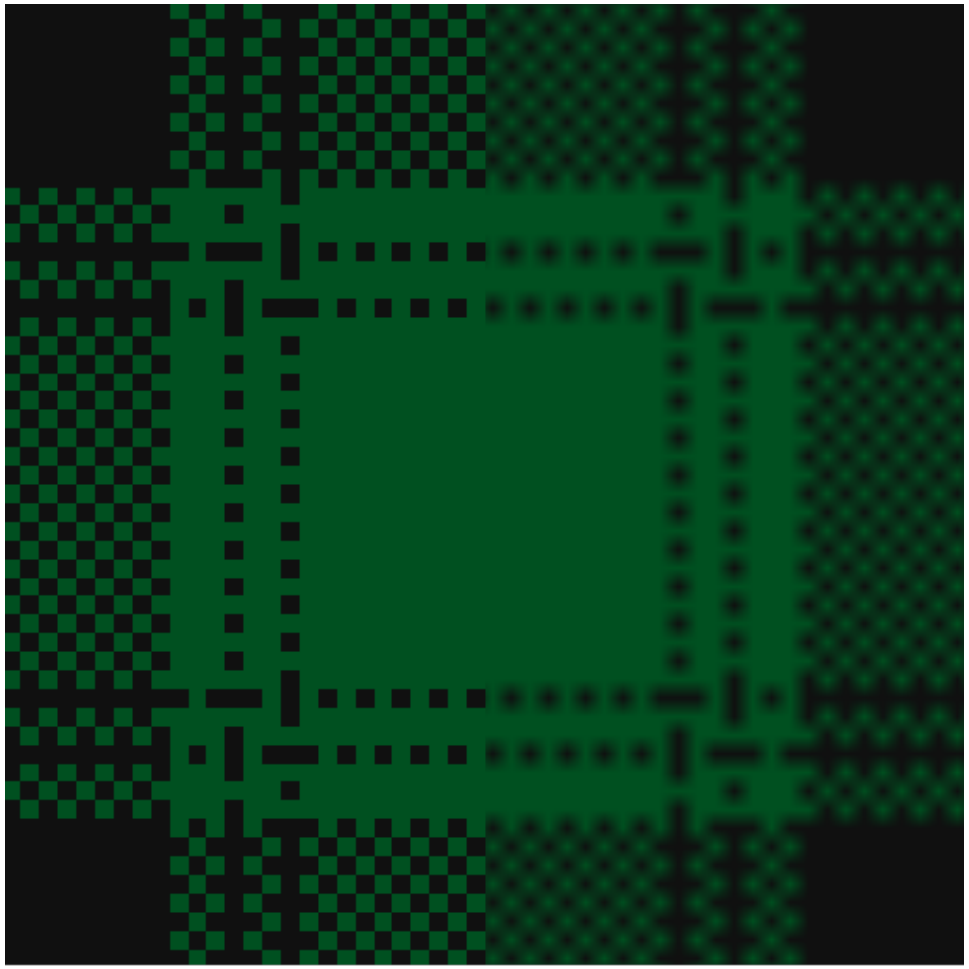


Practical Solution

- Give each vertex texture coordinates *s,t*
- Use barycentric interpolation on triangles
- Look up the resulting *s,t* in the image
  - As (row, col) coordinates
- But the coordinates are almost never integers
- So we will need interpolation



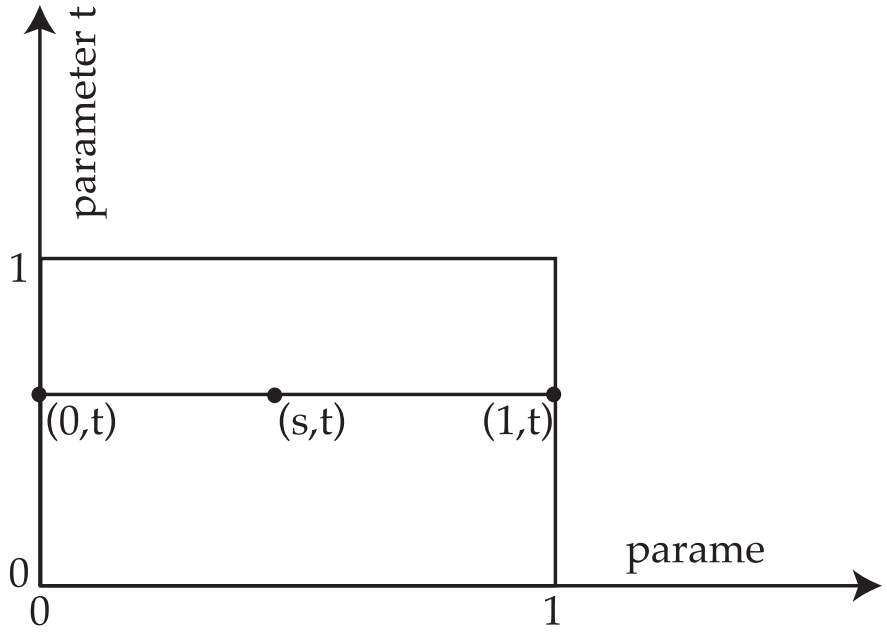
Texture Interpolation



Nearest Neighbour

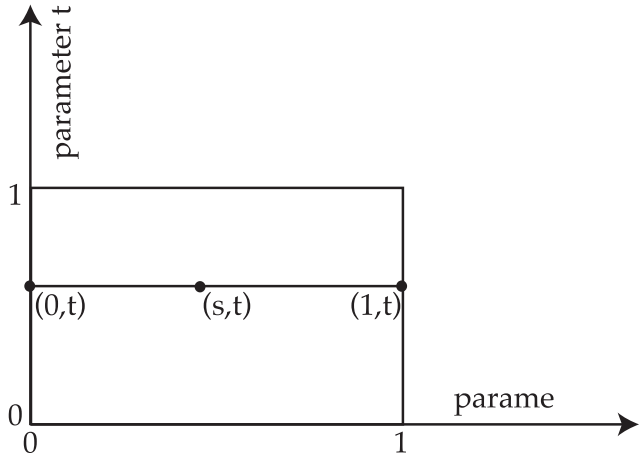
Bilinear

Bilinear Interpolation



- Interpolates on a quadrilateral
- Iterates linear interpolation twice
- Linear along the edges
- Commonly used for texture interpolation

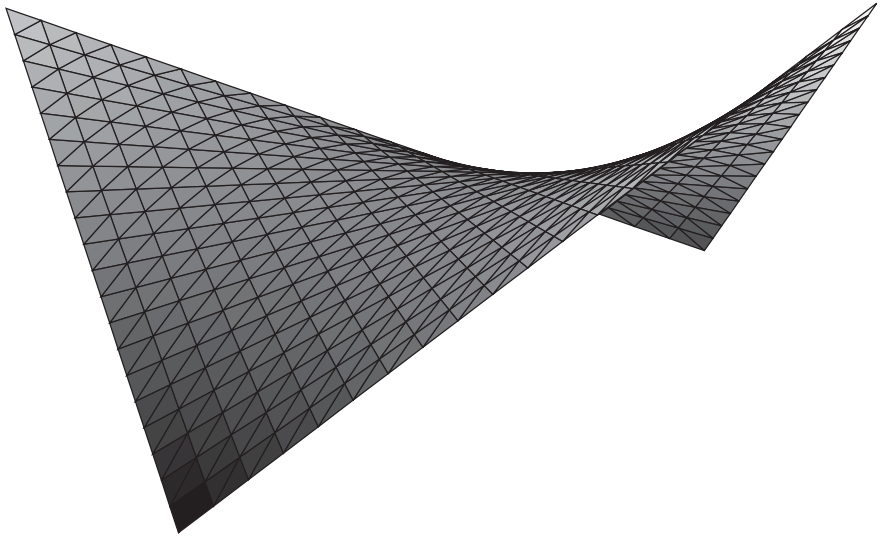
Development



$$
\begin{aligned}
b_{00} &= f(0,0) \\
b_{01} &= f(0,1) \\
b_{10} &= f(1,0) \\
b_{11} &= f(1,1)
\end{aligned}
$$

$$
\begin{aligned}
f(0,t) &= t \cdot b_{01} + (1-t) \cdot b_{00} \\
f(1,t) &= t \cdot b_{11} + (1-t) \cdot b_{10} \\
f(s,t) &= s \cdot f(1,t) + (1-s) \cdot f(0,t) \\
&= s \cdot (t \cdot b_{11} + (1-t) \cdot b_{10}) + (1-s) \cdot (t \cdot b_{01} + (1-t) \cdot b_{00}) \\
&= b_{11}st + b_{10}s - b_{10}st + b_{01}t - b_{01}st + b_{00} - b_{00}s - b_{00}t + b_{00}st \\
&= (b_{11} - b_{10} - b_{01} + b_{00})\, st + (b_{10} - b_{00})\, s + (b_{01} - b_{00})\, t + b_{00}
\end{aligned}
$$

Bilinear Surface



- We get a small height field
- Which is smooth / twisted
- So we'll see more of these later

Code

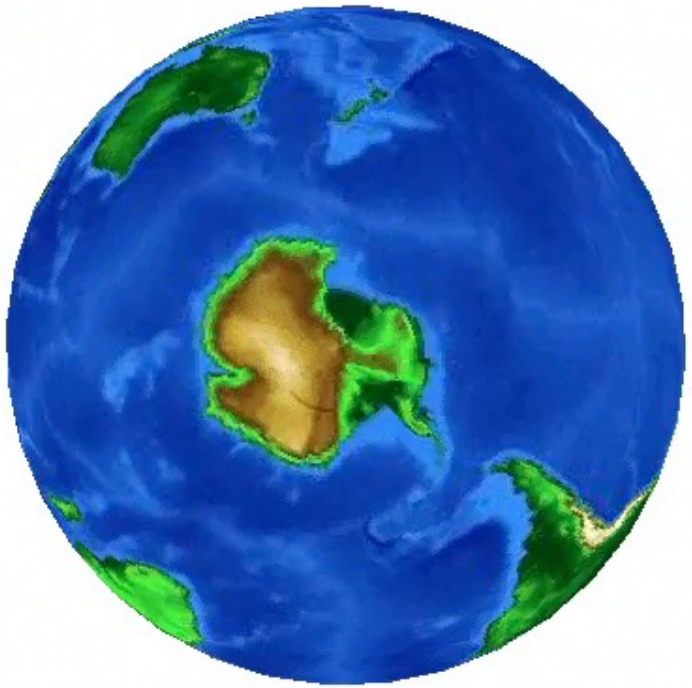
```
RGBValue BilinearLookup(Image tex, float s, float t)
  { // BilinearLookup()
  int i = s;                    // truncates s to get i
  int j = t;                    // truncates t to get j
  float sParm = s - i;          // compute s for interpolation
  float tParm = t - j;          // compute t for interpolation

  // grab four nearest texel colours
  RGBValue colour00 = tex[i][j];
  RGBValue colour01 = tex[i][j+1];
  RGBValue colour10 = tex[i+1][j];
  RGBValue colour11 = tex[i+1][j+1];

  // compute colours on edges
  RGBValue colour0 = colour00 + tParm * (colour01 - colour00);
  RGBValue colour1 = colour10 + tParm * (colour11 - colour10);

  // compute colour for interpolated texel
  return colour1 + sParm * (colour1 - colour0);
  } // BilinearLookup()
```


Bilinear Interpolation



Nearest Neighbour

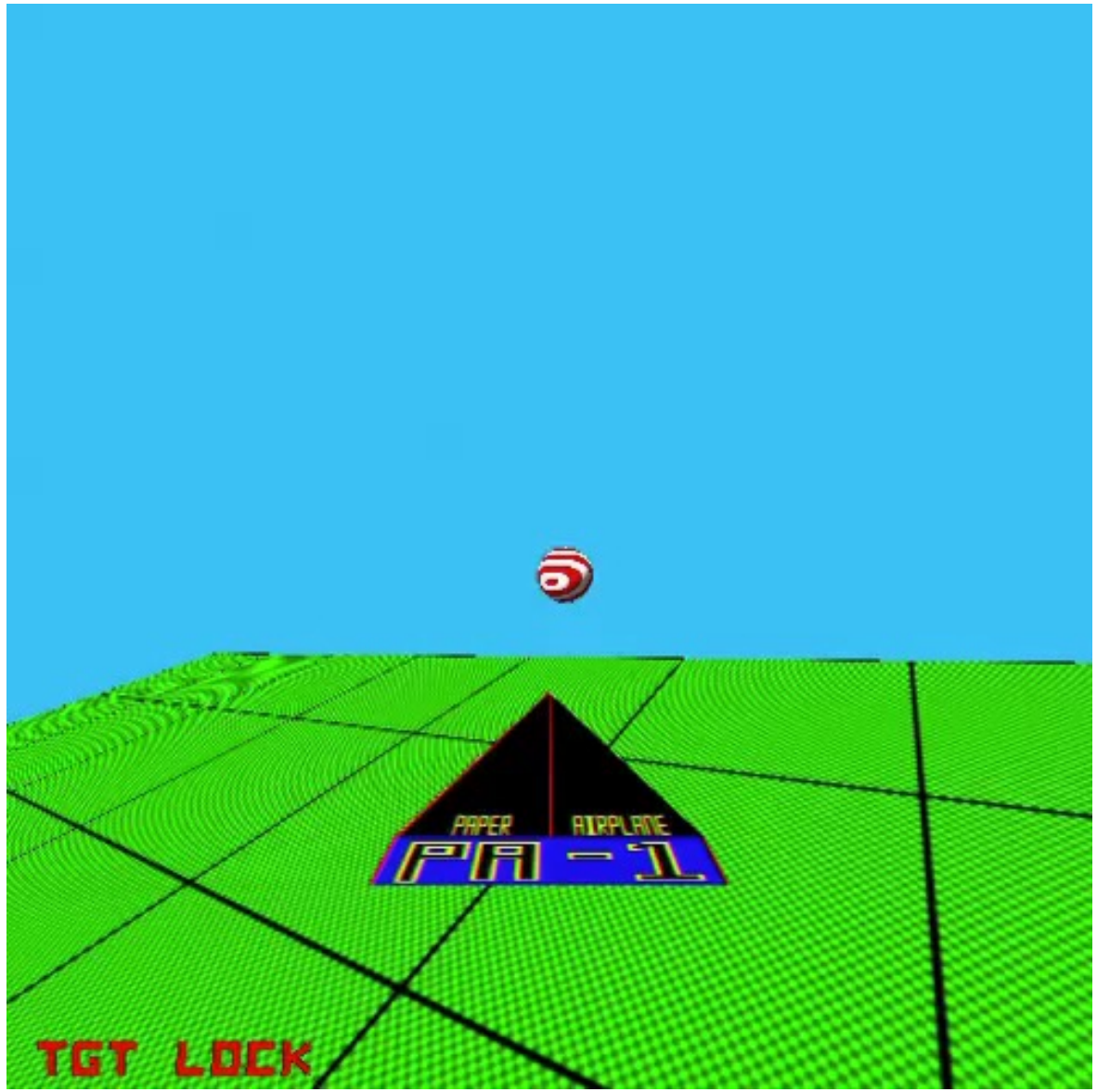
Bilinear

Replace vs. Modulate

GL_REPLACE

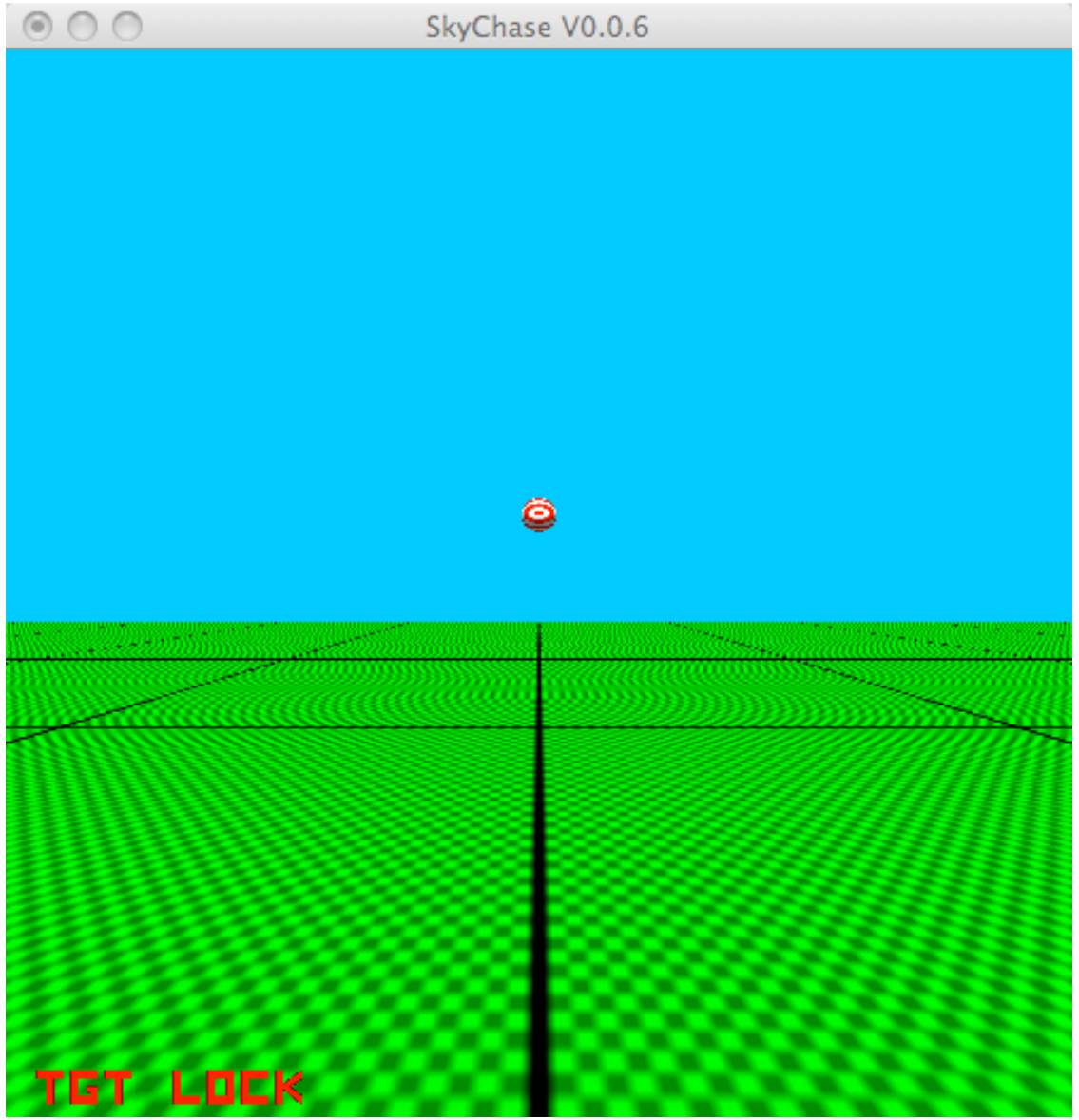
GL_MODULATE

But . . .



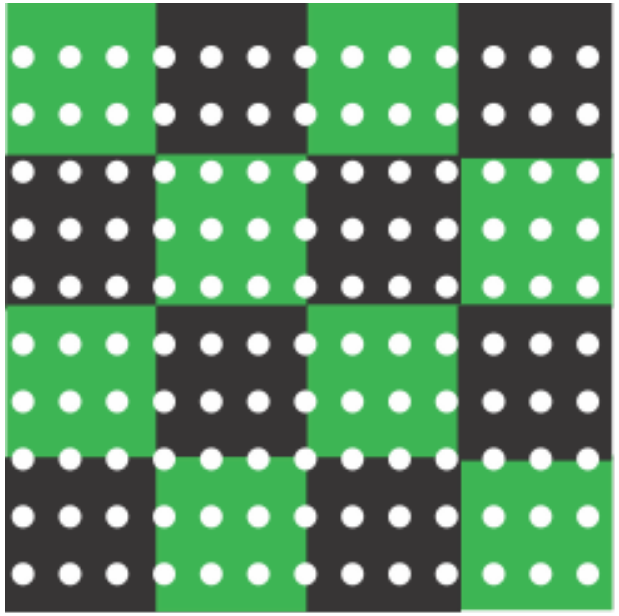
What Is Happening?

- Perspective *increases* background frequency
- No longer band-limited
- Result: Moire patterns / aliasing

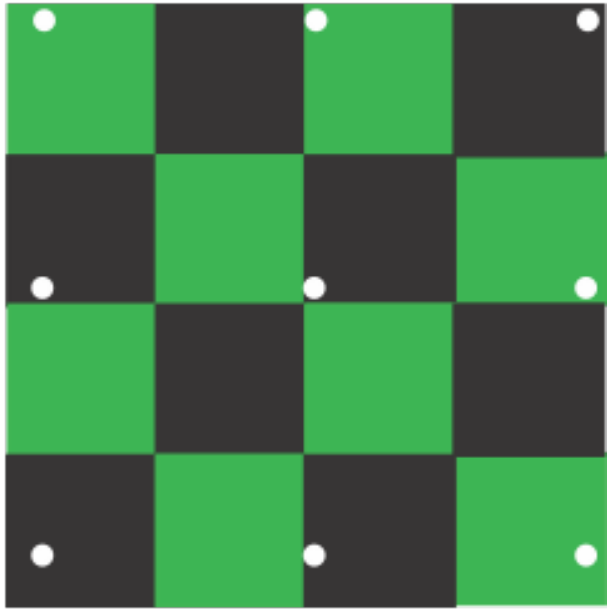


Point Sampling

- Point sampling is best if it's *dense*
- Perspective causes frequency to decrease



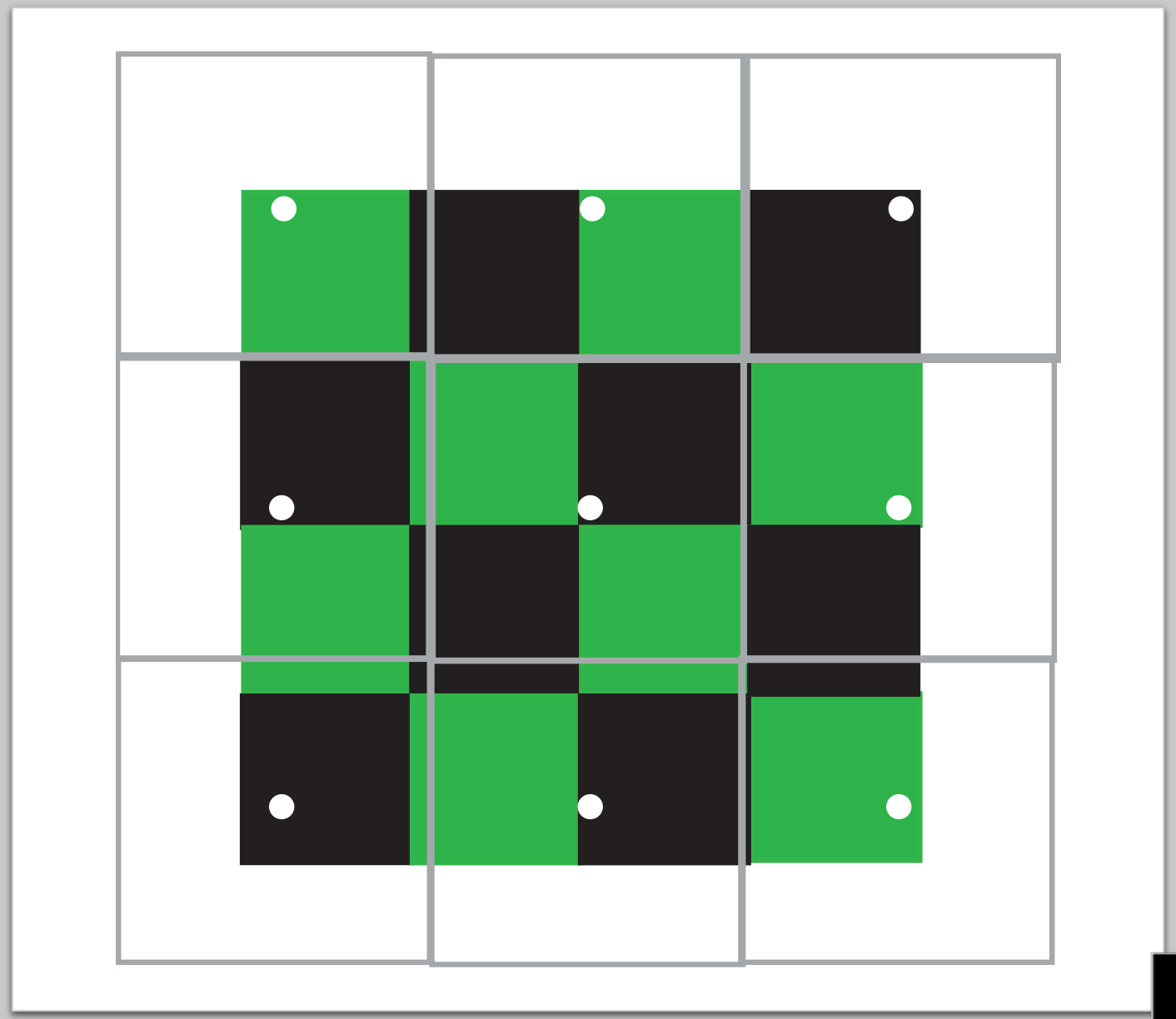
Near the Eye



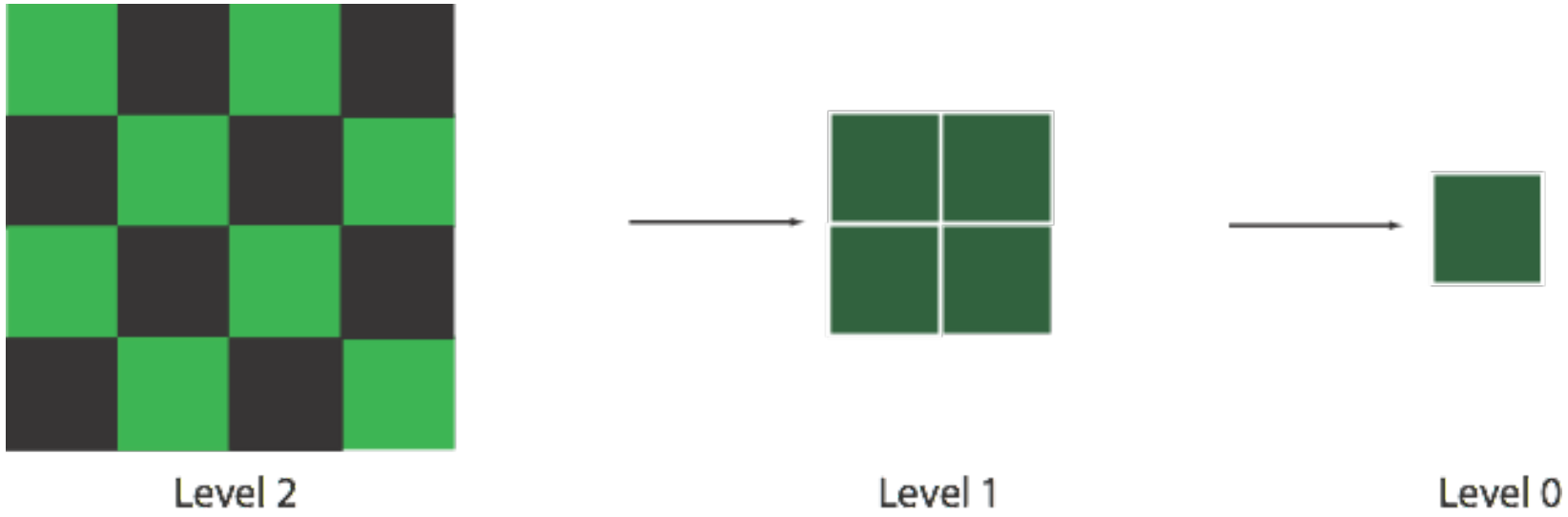
Far From the Eye

Correct Solution

- Integrate the light over patches
- But this wastes samples on background



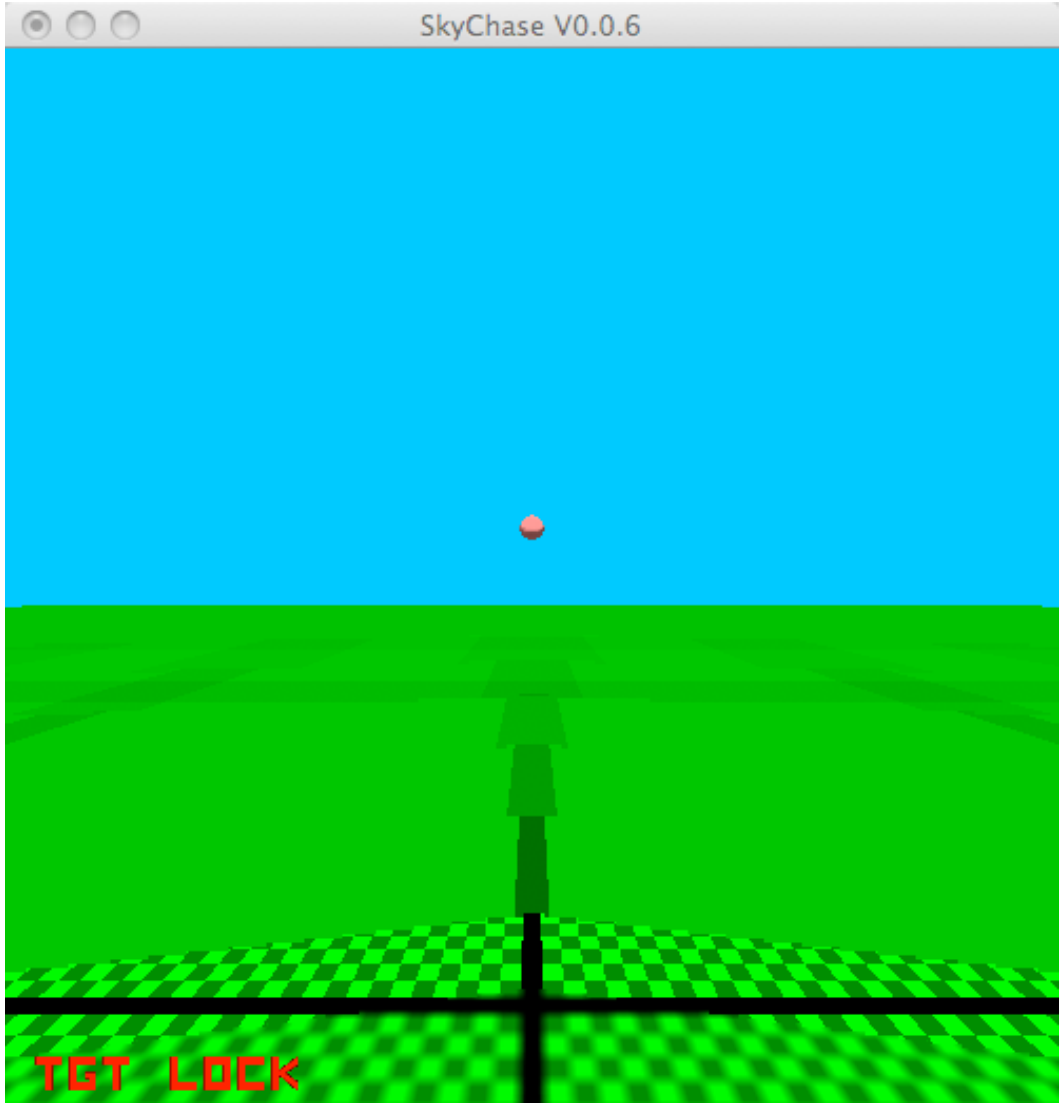
Approximation



- Compute patches at half resolution
- Repeat recursively
- *Mipmaps* = multum in parvum maps

Improvement?

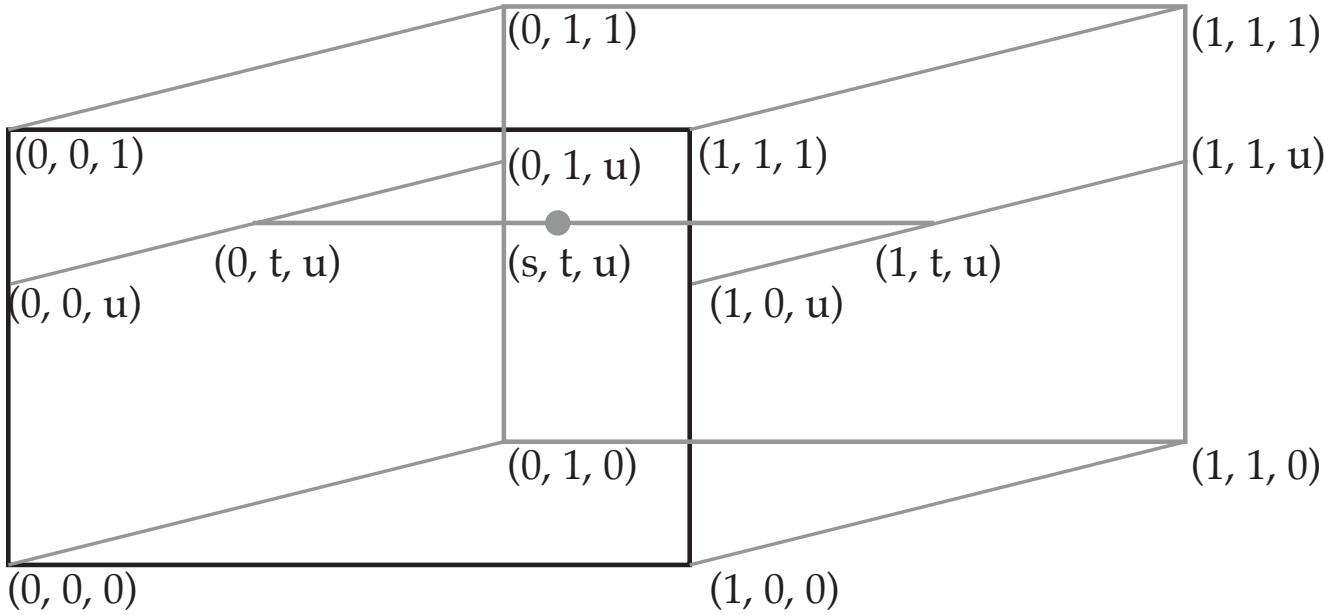
- Choosing transition range is important
- And you never *quite* get it right
- So how to make it less abrupt?



Trilinear Mipmapping

- Identify two ranges (distances) *near, far*
  - $near < distance < far$
  - Parametrise pixel distance as:
    - $dist\_parm = \frac{distance - near}{far - near}$
- Use bilinear interpolation twice
- Then interpolate between the two colours
  - Result: trilinear interpolation

Trilinear Interpolation



- Same idea as before, but repeated three times
- Development is similar

Trilinear Equation

$$f(x,y,z) = a\ xyz + b\ yz + c\ xz + d\ xy + ex + fy + gz + h$$

$$a = b_{111} - b_{110} - b_{101} - b_{011} + b_{100} + b_{010} + b_{001} + b_{000}$$

$$b = b_{011} - b_{010} - b_{001} + b_{000}$$

$$c = b_{101} - b_{100} - b_{001} + b_{000}$$

$$d = b_{110} - b_{100} - b_{010} + b_{000}$$

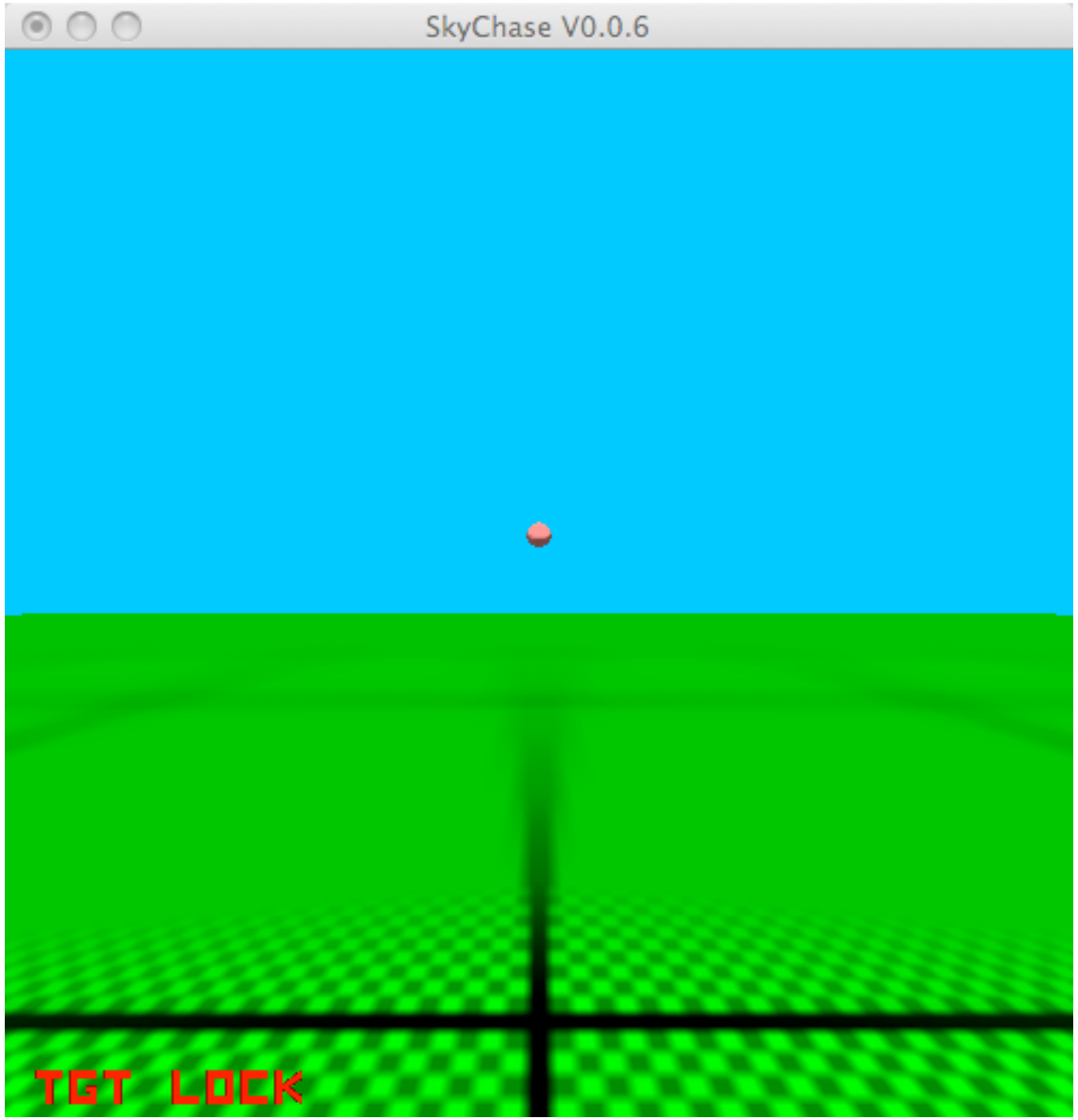
$$e = b_{100} - b_{000}$$

$$f = b_{010} - b_{000}$$

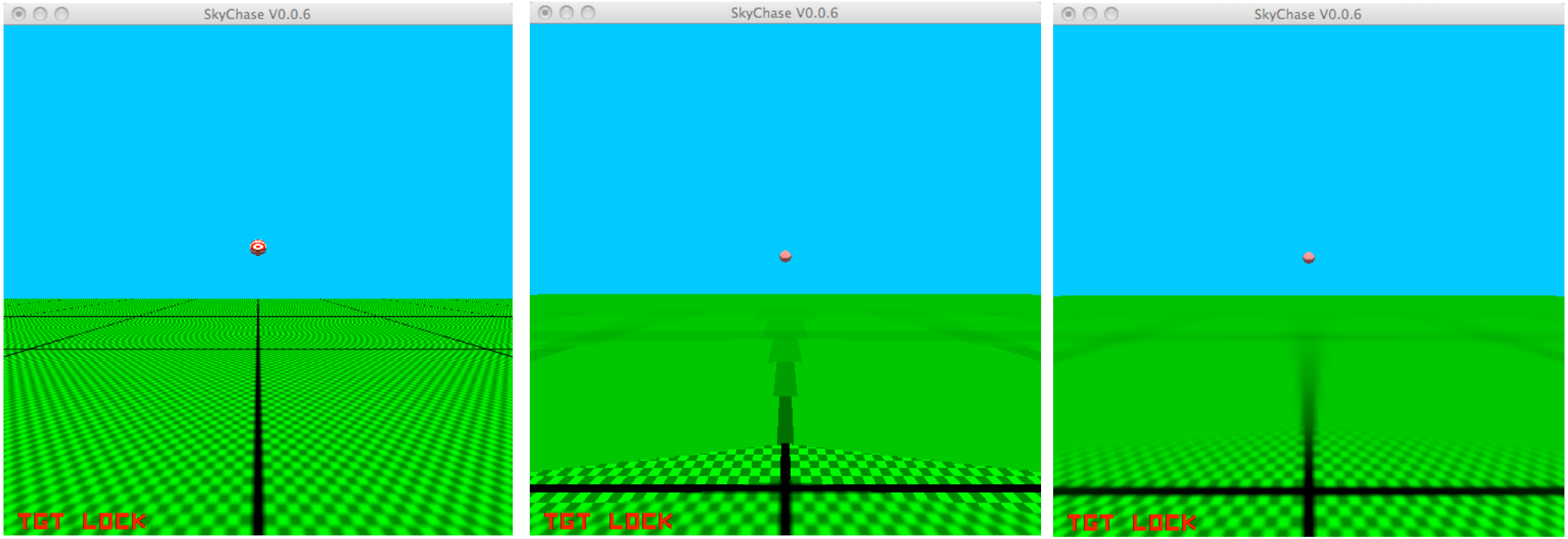
$$g = b_{001} - b_{000}$$

$$h = b_{000}$$

Trilinear Mipmapping



Comparison

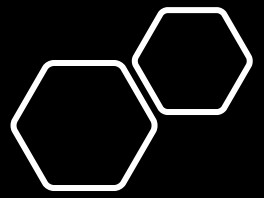

SkyChase V0.0.6
TGT LOCK
SkyChase V0.0.6
TGT LOCK
SkyChase V0.0.6
TGT LOCK

More generally

- Mipmapping is a special case
- We need a more general solution
- But scaling up & scaling down are different
  - And this requires some image processing

Example of Mappable Properties

- Normal vectors
- Colours
- Material properties
- BSSDFs
- Surface deformations
- Random seeds (use Nearest Neighbour)
- Coefficients for calculations



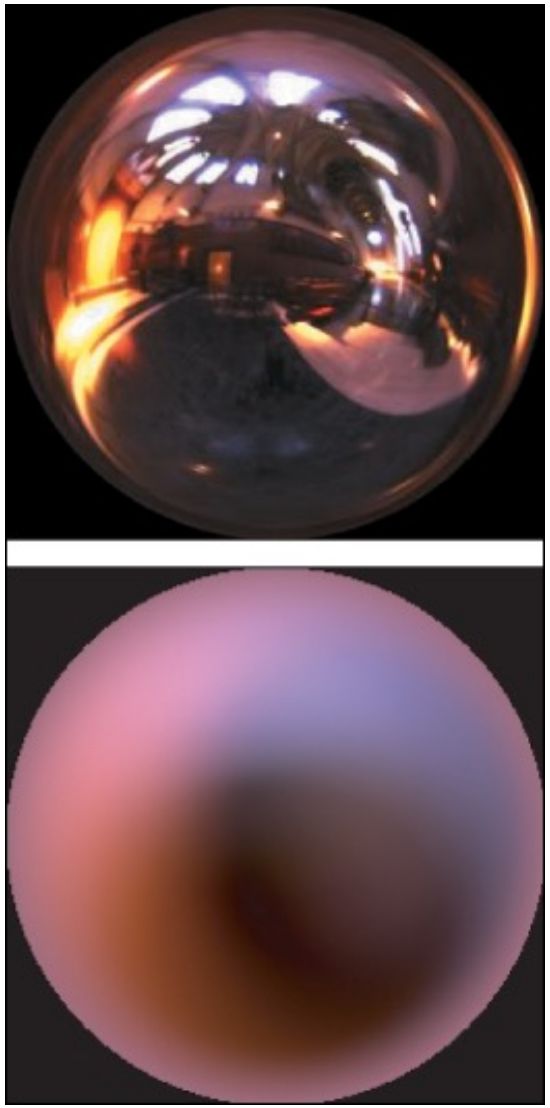
In the Phong Model

- Diffuse reflection constant
- Specular reflection coefficient
- Diffuse, specular, ambient, emissive colours
- Specular exponent
- Whether or not in a shadow
- Extra parameters for lighting equations
- All of these can be made (u,v)-dependent

Environment Maps

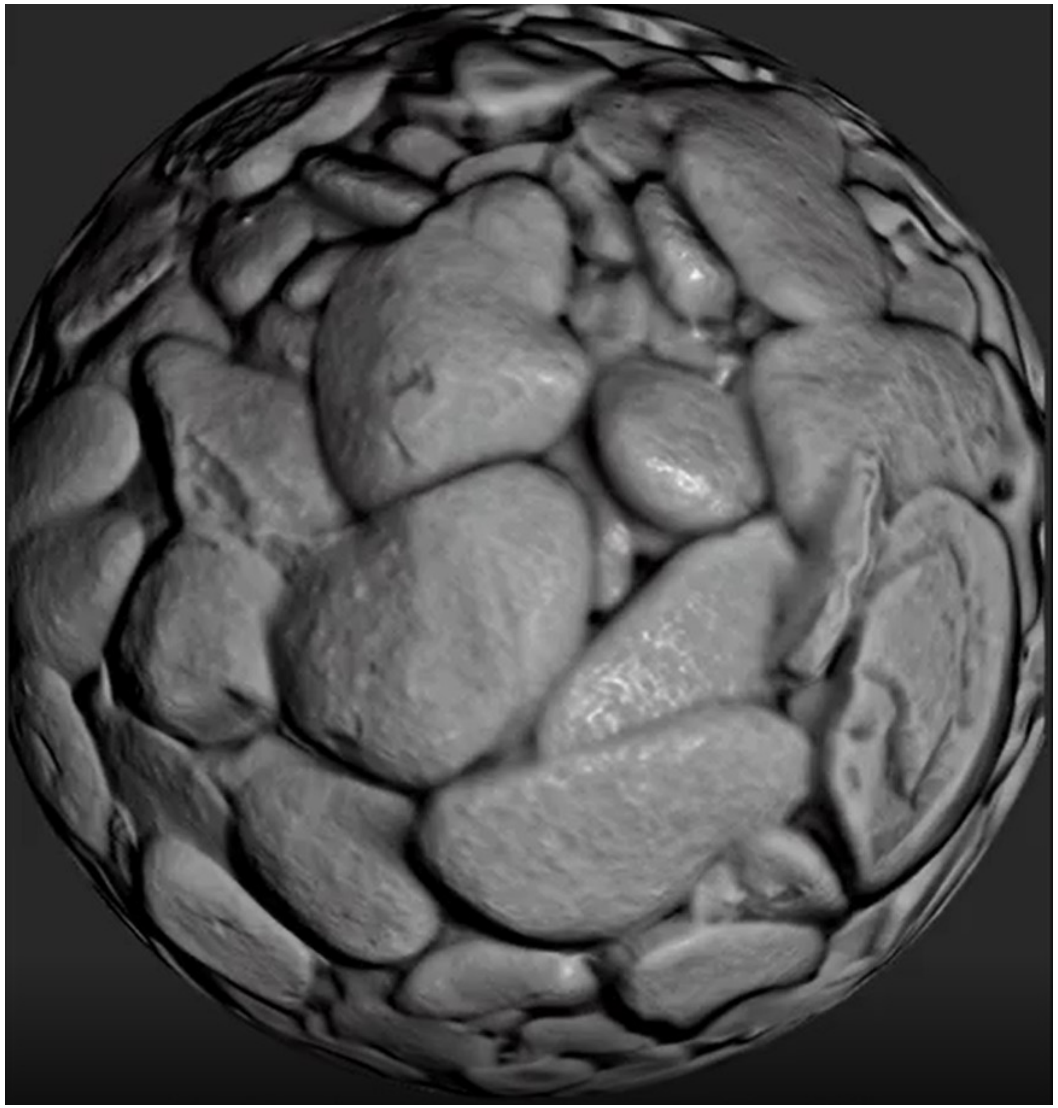
Irradiance Mapping

- Diffuse surfaces are a different summation
- So precompute the lighting on them
- Approximates room lighting
- Apply to multiple objects



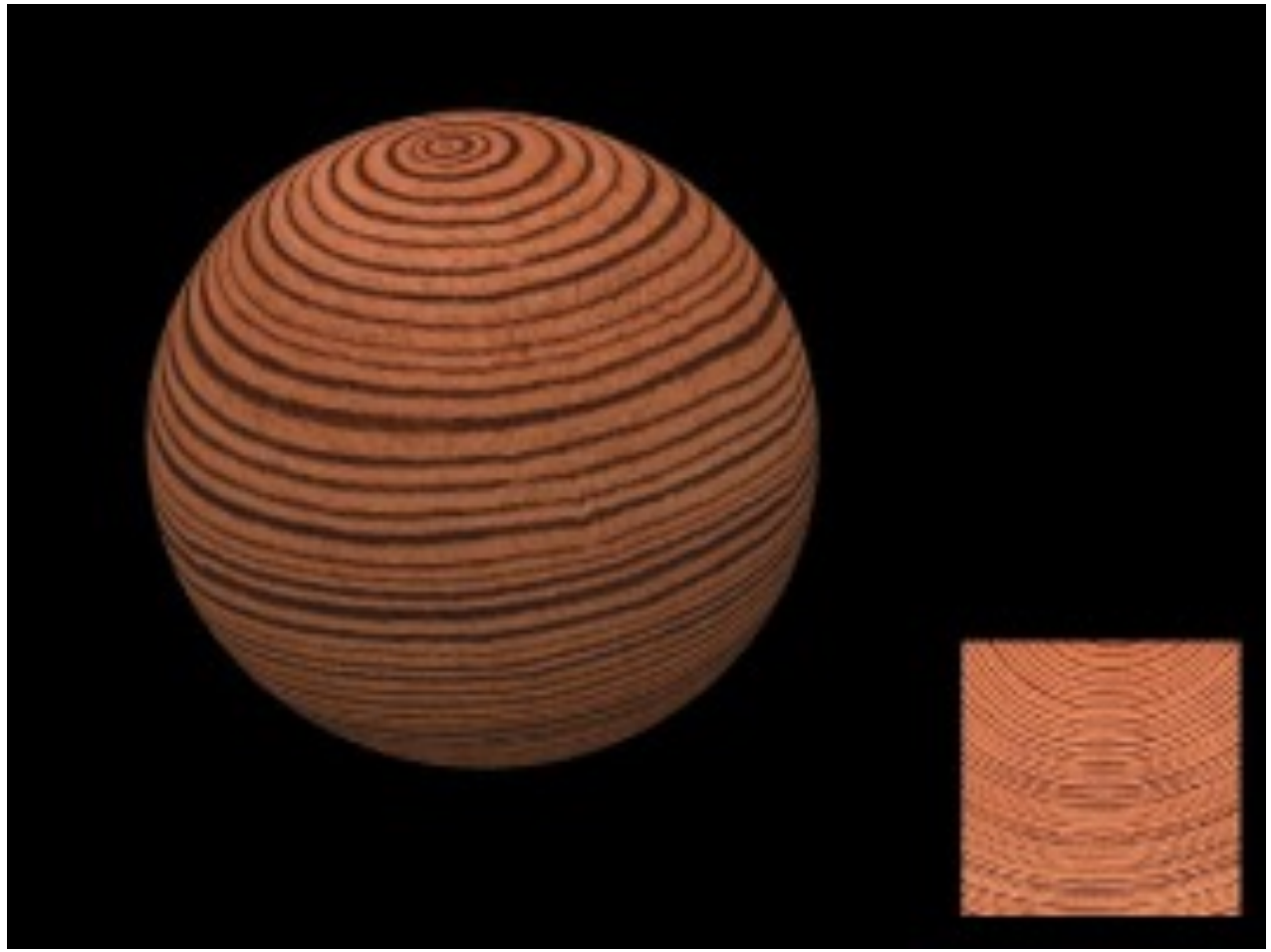
Bump Mapping

- Models small changes in normal vectors
- Simulates shadows from surface roughness
- Stores per-pixel normal vectors
  - i) convert *(x,y,z)* to *(r,g,b)*
  - ii)normal perturbation stored as *(r,g)*
  - iii) height field stored as *(r)*



Volumetric Contours

- Why not *3* coordinates?
- Specify position in space
- Use that to look up texture
- Result: marble/wood/cheese
- But volumetric textures are expensive

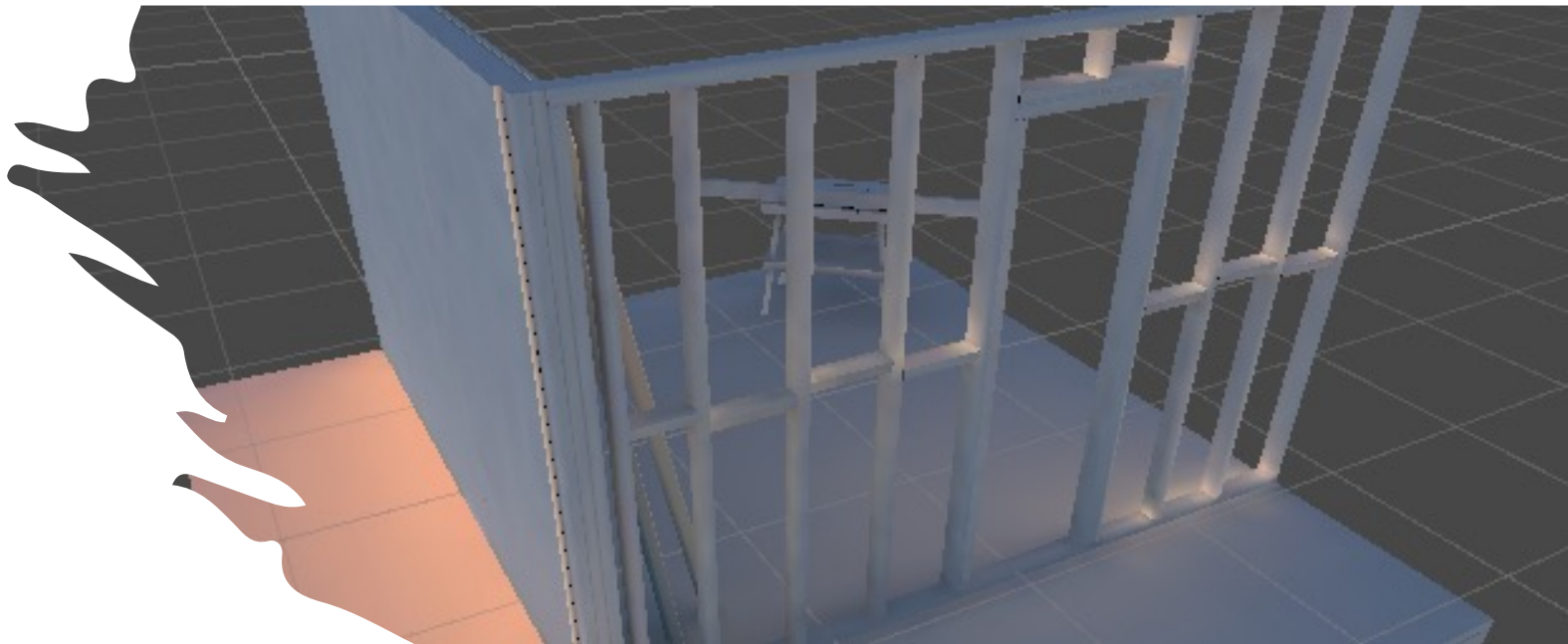


Video Textures

- Store each frame as a slice in 3D
- Store (u,v) coordinates for the screen
- Set w = t based on the *time* you render
- Result – video plays back on surface

Light Maps

- Precompute lighting from each light source
- Store in separate textures
- Compute final lighting as sum of active lights
- Ignores characters in the scene
- But massively speeds up scenery rendering
- Leads to shadow maps (next term)



Projective Lighting

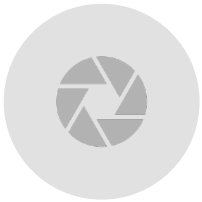
- Light coming *through* a translucent surface
- Store the light colour in a texture map

How do you design Textures?

PICTURES (E.G. ENVIRONMENT MAP)

ACQUIRED (SCANNING DEVICES)

ARTIST-DESIGNED (PAINTED ON)



SPHERICAL PROJECTION

TEXTURE PARAMETRIZATION / DEFORMATION

TEXTURE SYNTHESIS

Creating Textures from attributes

Given a mesh M with

Vertex positions P	Texture coordinates U	Attribute values A

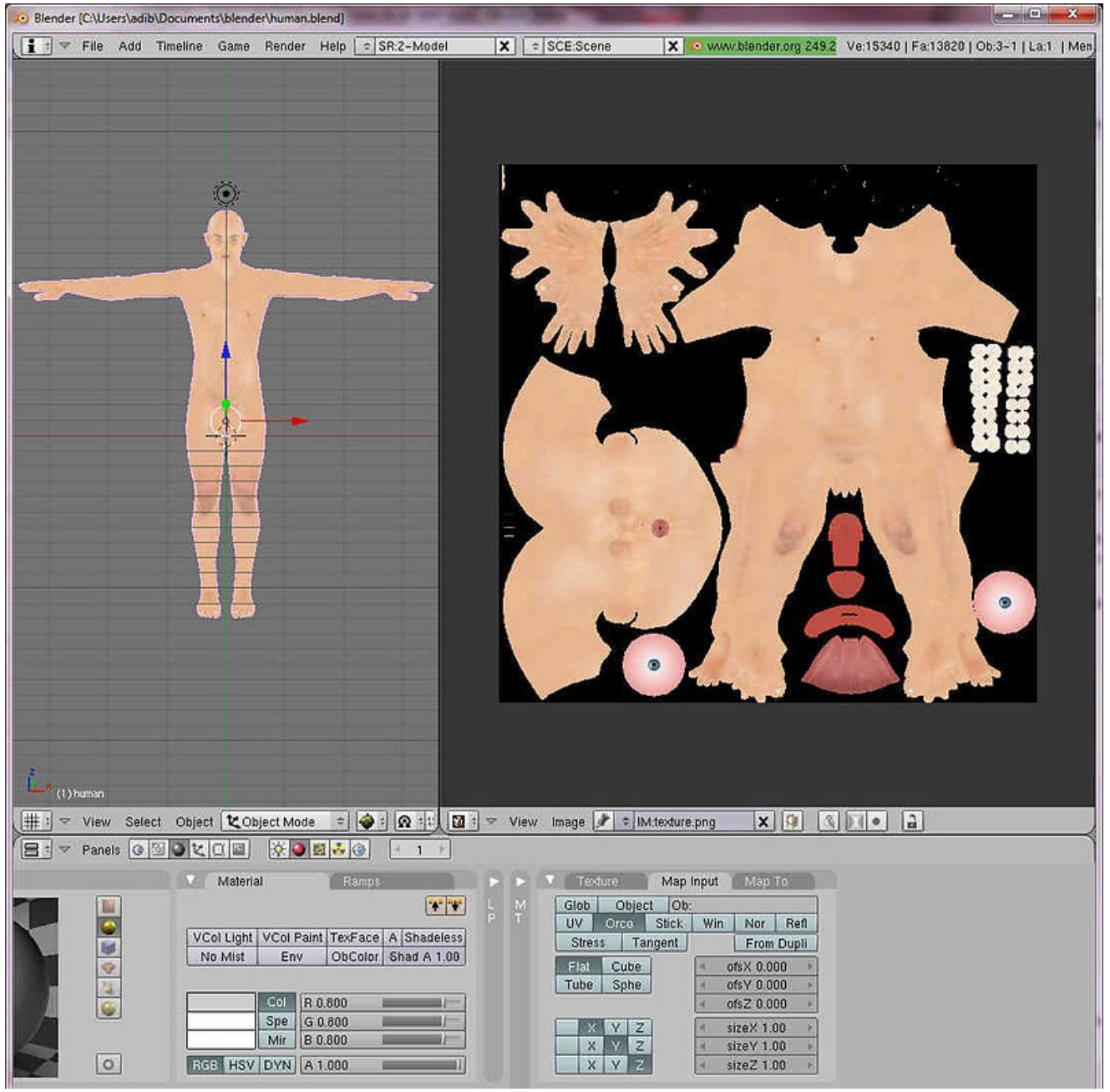
Render M in 2D

Using U as vertex positions, not P

And the texture will hold the attribute

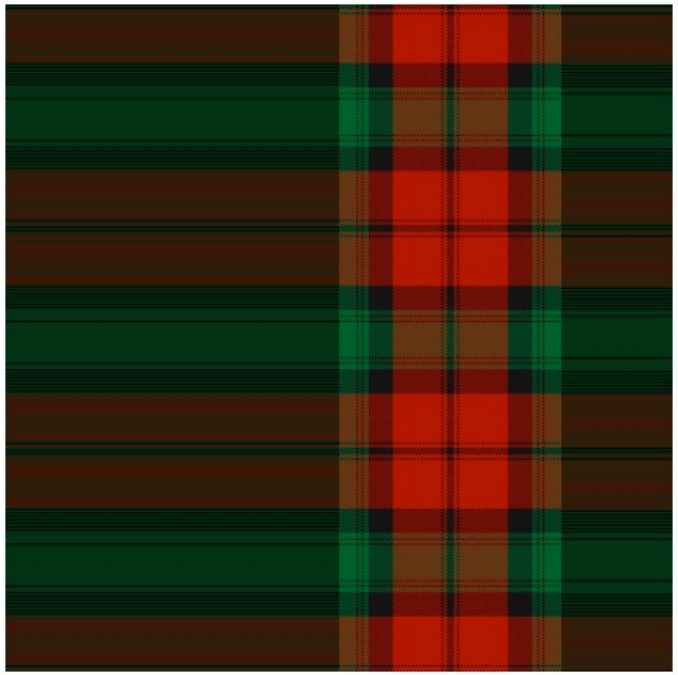
Artist Designed

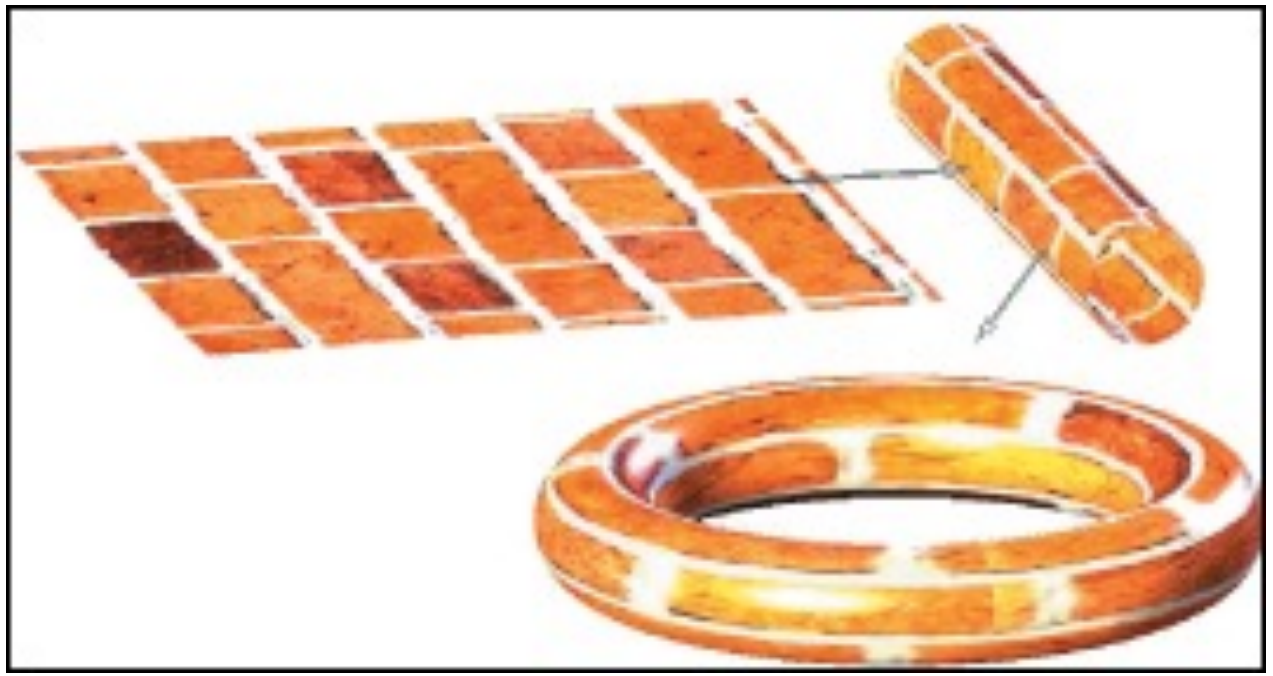
- Painted onto a model, uv coords set by hand



Othman Ahmad - screenshot CC BY-SA 3.0

Texture Boundaries





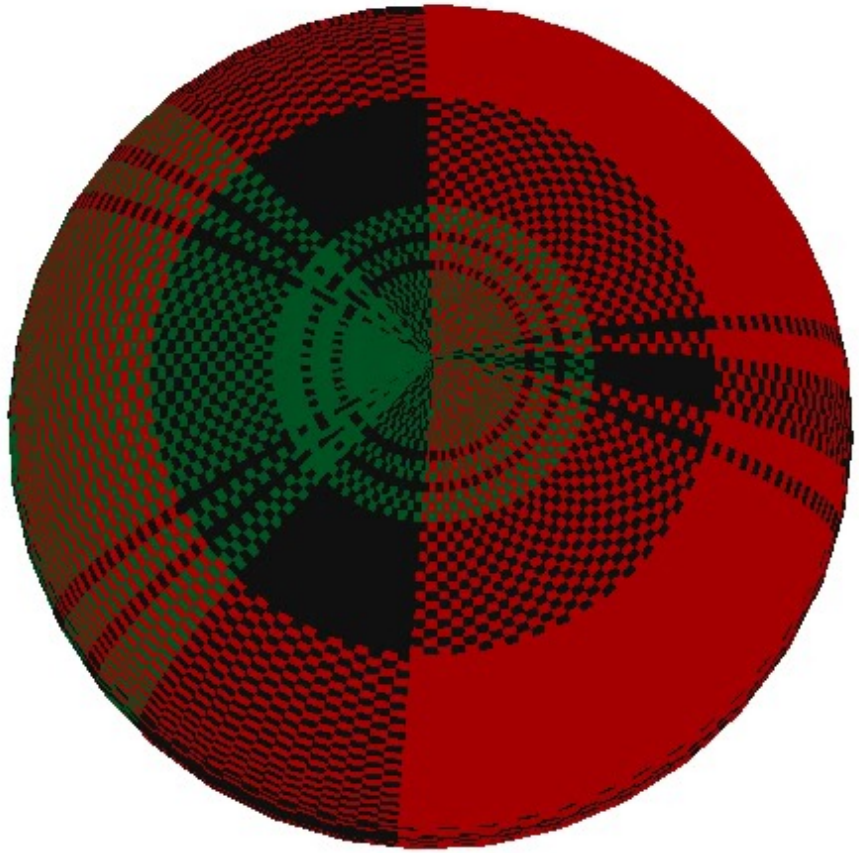
- What do we do at the *edge* of the texture?
- We know that we can clamp or repeat
- Repeating works for a torus or flat sheet

Texture Seams

- Most surfaces are *genus 0*
- A torus is *genus 1*
- Which means we can't just use repeat
- And we end up with seams in the textures
- Or map (project) each surface to a sphere
- Which we can cover with a single texture

Texture Distortion

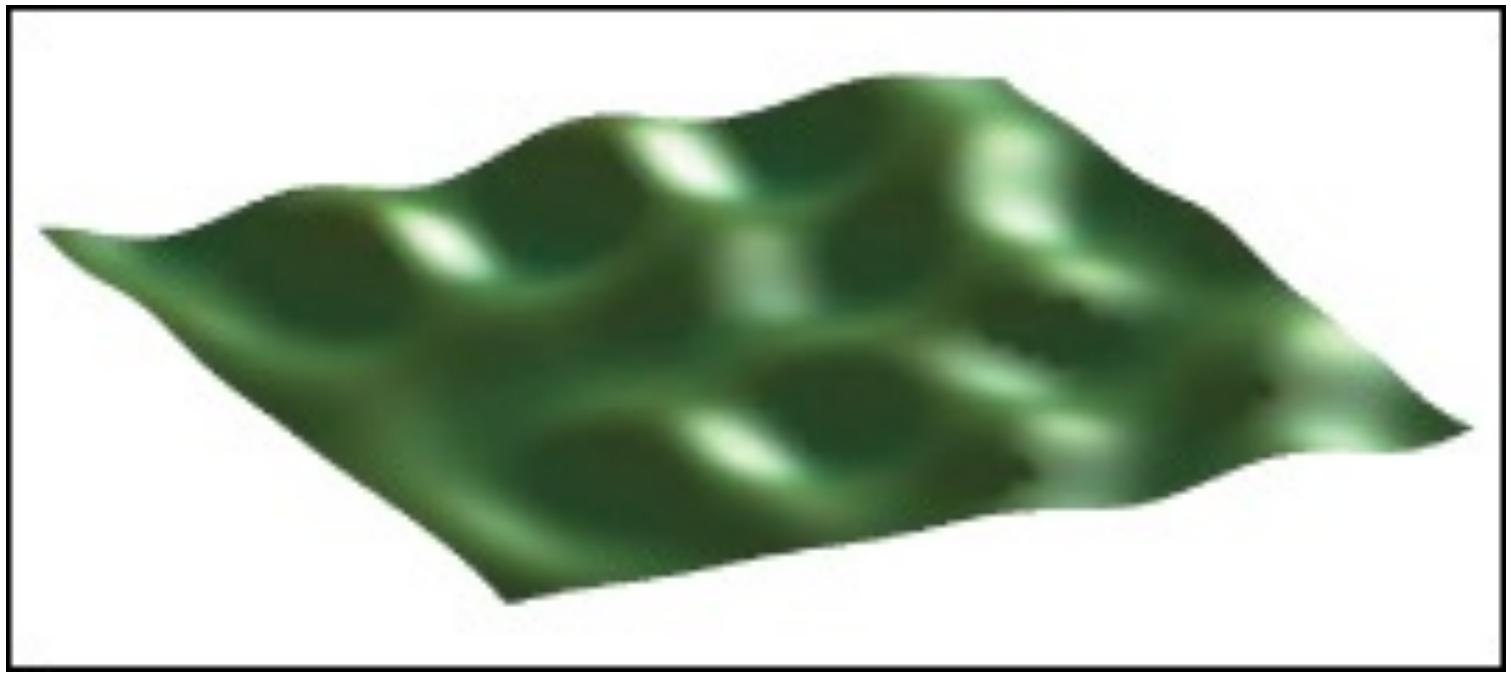
- Texture details are unevenly distributed
- How do we use texels *efficiently?*



Texture Synthesis

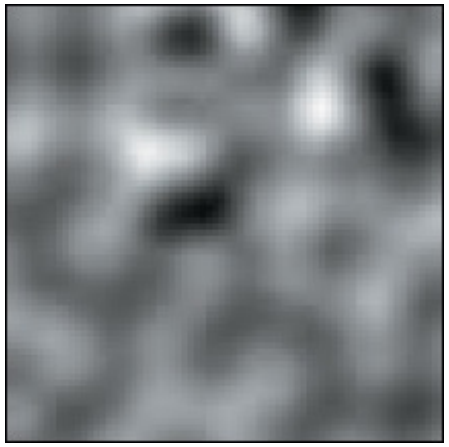
- Write code to create texture
- Not good for humans
- Pretty good for some tasks
  - Fourier Synthesis
  - Perlin Noise
  - Reaction-Diffusion
  - Data-Driven

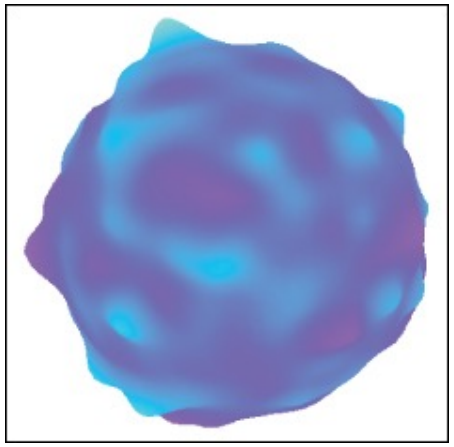
Fourier Synthesis

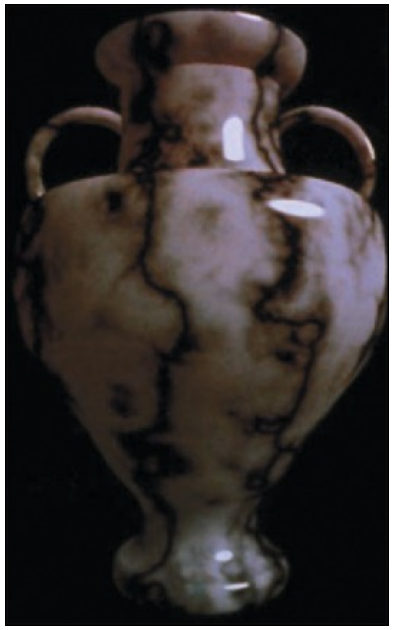


- Define a sum of sines & cosines
- Use that for a mottled appearance

Perlin Noise





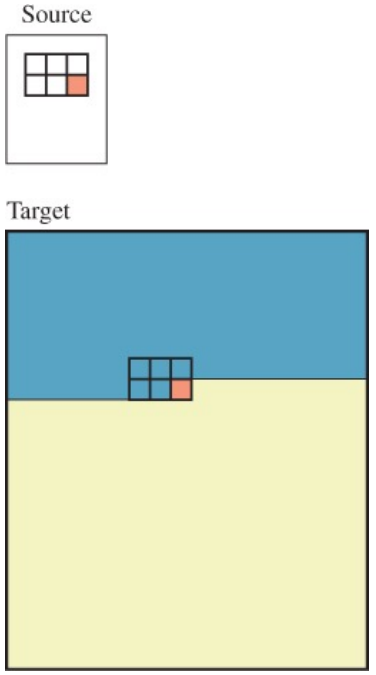


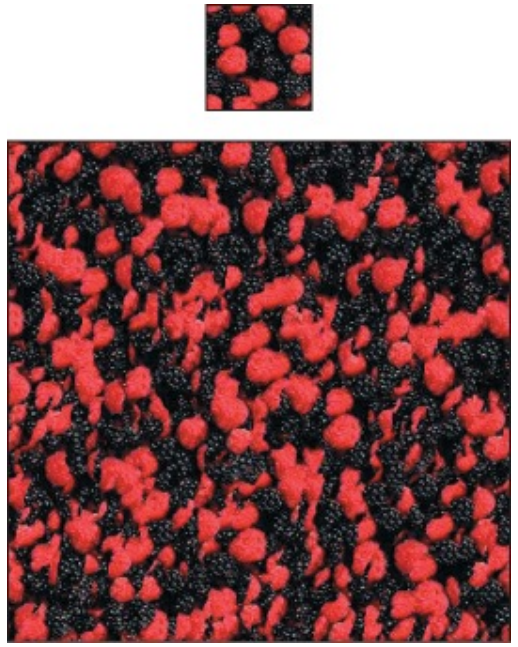
- Randomise some points
- Use spline curves / surfaces to interpolate
- Use this to generate an image

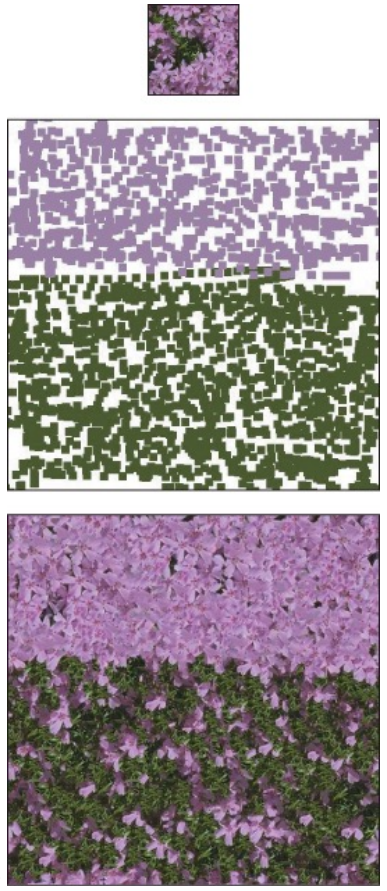
Reaction-Diffusion

- Models pigment production in skin
- Useful for animal coats, &c.
- Essentially a small numerical simulation

Data-Driven Generation







- Use a small travelling window
- Pick a matching window in sample image
- Copy next pixel based on that

Image (Re-)Synthesis

- Huge overlap with image processing

Summary

- Texture mapping is an indirection
- Allows mapping *any* property to surface
- Therefore used as a lookup mechanism
- Either in an array or in a function
- Many, many variations
- Geometry underneath drives a lot of it