Real-world surfaces are colorful!

photo from K.C. Alfred
An option: assign a color to each triangle

• Q: pros and cons?
An option: assign a color to each triangle

• pros
  • simple
  • easy to edit (just paint on triangles)

• cons
  • couples geometric complexity with color complexity
  • hard to filter
In practice: texture mapping

- assign a “UV” 2D vector to each point on the surface
Observe: the UV may land on texture pixel ("texel") center

(image from pbrt-v2 pbrt.org/scenes-v2)
We need to interpolate the texels

usually done using bilinear interpolation

\[ f(u, v) = (1 - \Delta_u)(1 - \Delta_v)F_{00} + \Delta_u(1 - \Delta_v)F_{10} + (1 - \Delta_u)\Delta_vF_{01} + \Delta_u\Delta_vF_{11} \]

https://thebookofshaders.com/11/
UV mapping

• “unwrap” a surface and map it to a 2D square

• automatic UV mapping is an active research area

http://staff.ustc.edu.cn/~fuxm/projects/Peeling/index.html
UV mapping

- usually assign a UV value per vertex
- interpolate between them using barycentric coordinates

\[ uv = b_0 uv_0 + b_1 uv_1 + b_2 uv_2 \]
Distortion is often unavoidable in UV mapping

- ideally we want
  
  - area-preserving (large areas map to large areas)
  
  - conformal (angles between any two curves are preserved)
  
- a theorem from Euler [1775]:
  
  - for a sphere-to-square projection, a conformal map cannot be area-preserving
  - an area-preserving map cannot be conformal

https://en.wikipedia.org/wiki/Mercator_projection
An area-preserving projection of earth
A few simple “automatic” UV mappings

spherical mapping
cylindrical mapping
planar mapping
Spherical mapping

(θ, φ)
Cylindrical mapping

$$(\theta, r)$$
Planar mapping

\((i, u, v)\)
3D Textures

use a volumetric texture, don’t need UV mapping then

An option: assign a color to each triangle

- **pros**
  - simple
  - easy to edit (just paint on triangles)

- **cons**
  - couples geometric complexity with color complexity
  - hard to filter
A pixel can cover a large region in the texture
Fail to account for all texels in the region can lead to noise/aliasing

image from Steve Marschner
Averaging over all texels in a region can be slow.
Mipmapping: build an image pyramid for fast averaging
Tri-linear interpolation

interpolate not just between the texels, but also between mipmap levels
In practice, estimate the size of region using derivative between \((x, y)\) & \((u, v)\)

see “ray differentials” https://graphics.stanford.edu/papers/trd/
Comparison: point sampling

image from Steve Marschner
Comparison: 512x anti-aliased
Comparison: 1 pixel sample with mipmapping

image from Steve Marschner
In graphics hardware

in fragment shader
(mipmapping is also done in hardware!)

out vec4 FragColor;

in vec3 ourColor;
in vec2 TexCoord;

uniform sampler2D ourTexture;

void main() {
    FragColor = texture(ourTexture, TexCoord);
}

see here to learn more
https://learnopengl.com/Getting-started/Textures
Now you know what these mean!
Procedural textures: textures as programs

def stripe(x, y, z, u, v):
    if int(x) % 2 == 0:
        return red
    else:
        return white
Procedural textures: textures as programs

```python
def ramp(x, y, z, u, v):
    v = (sin(x) + 1) / 2
    return (1 - v) * magenta +
           v * yellow
```
Procedural textures: textures as programs

```python
def ring(x, y, z, u, v):
    v = (x - center.x)^2 + (y - center.y)^2
    if int(v) % 2 == 0:
        return white
    else:
        return red
```

https://www.csie.ntu.edu.tw/~cyy/courses/rendering/16fall/lectures/handouts/chap10_textures.pdf
Procedural noise want to represent random natural variation
Procedural noise

want to represent random natural variation
Perlin noise

• a way to procedurally generate stochastic textures

• used in TRON (1982)!
  (first Hollywood film that used 3D shaded graphics)
Idea: smoothly interpolate white noise
Perlin’s Noise function

```python
def Noise1D(x):
    xi0 = int(x)
    xi1 = xi0 + 1
    val0, val1 = hash(xi0), hash(xi1)
    t0, t1 = x - xi0, xi1 - x
    w0 = 6 * t0^5 - 15 * t0^4 + 10 * t0^3
    w1 = 6 * t1^5 - 15 * t1^4 + 10 * t1^3
    return w0 * val0 + w1 * val1
```

see Perlin [2002] for the weight derivation
Perlin’s Noise function

```python
def Noise2D(x):
    xi0, yi0 = int(x), int(y)
    xi1, yi1 = xi0 + 1, yi0 + 1
    val00, val01 = hash(xi0, yi0), hash(xi0, yi1)
    val10, val11 = hash(xi1, yi0), hash(xi1, yi1)
    t0x, t1x = x - xi0, xi1 - x
    t0y, t1y = y - yi0, yi1 - y
    w00 = w(t0x) * w(t0y)
    ...
    return w00 * val00 + ...
```

see Perlin [2002] for the weight derivation

https://thebookofshaders.com/11/
Scale of the inputs determines the smoothness of the noise

\[ \text{Noise2D}(x/n) \]
Combining multiple Noises at different scales

\[ \text{Noise1D}(x) + \frac{1}{2} \times \text{Noise1D}(2x) + \frac{1}{4} \times \text{Noise1D}(4x) + \ldots \]
Combining multiple Noises at different scales
Turbulence

\[ \frac{1}{2} \cdot |\text{Noise3D}(2x)| + \frac{1}{4} \cdot |\text{Noise3D}(4x)| + \ldots \]
Marble

perturb uv using turbulence
Map generation

- high -> green
- middle -> brown
- low -> blue

https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401
Map generation

- high -> white
- mid-high -> gray
- middle -> green
- mid-low -> brown
- low -> blue

https://medium.com/@yvanscher/playing-with-perlin-noise-generating-realistic-archipelagos-b59f004d8401
Inverse Perlin noise

target image

optimization

2021
Adding details to smoke simulation using Perlin’s turbulence noise

(won an Oscar technical achievement award at 2012!)

Wavelet Turbulence for Fluid Simulation

Theodore Kim    Nils Thürey    Doug James    Markus Gross

2008
A great resource on noise

The Book of Shaders
by Patricio Gonzalez Vivo and Jen Low

This is a gentle step-by-step guide through the abstract and complex universe of Fragment Shaders.

https://thebookofshaders.com/11/
Next: lighting