Computer Graphics

- Texturing Methods -
Overview

• **Last time**
  – BRDFs
  – Shading

• **Today**
  – Texturing
    • Texture parameterization
  – Procedural methods
    • Procedural textures
    • Fractal landscapes

• **Next lecture**
  – Texture filtering
  – Alias & signal processing
TEXTURING
Simple Illumination

- No illumination
- Constant colors

- Parallel light
- Diffuse reflection
Standard Illumination

- Parallel light
- Specular reflection

- Multiple local light sources
- Different BRDFs

- Object properties constant over surface
Texturing

• **Varying object properties**
  – 2D image reflectance textures
  – Bump-mapping

• **Environment characteristics**
  – Shadows
  – Reflection textures
Texture-Modulated Quantities

- **Modulation of object surface properties**
  - Reflectance
    - Diffuse reflection coefficient $k_d$
    - Specular reflection coefficient $k_s$
  - Opacity ($\alpha$)
    - Modulating transparency (e.g. for fences)
  - Normal vector
    - Bump mapping: $N'(P) = N(P + t N)$ (in normal direction, height)
    - Normal mapping: $N' = N + \Delta N$ (arbitrary offset)
  - Geometry
    - Displacement mapping: $P' = P + \Delta P$

- **Distant illumination**
  - Environment mapping / reflection mapping
2D Texture Mapping

- **Forward mapping**
  - Object surface parameterization
  - Projective transformation

- **Inverse mapping**
  - Find corresponding pre-image/footprint of each pixel in texture
  - Integrate over pre-image
Forward Mapping

- Maps each texel to its position in the image
- Uniform sampling of texture space does not guarantee uniform sampling in screen space
  - Can create holes, need to scan-convert (see later)
- Possibly used if
  - The texture-to-screen mapping is difficult to invert
  - The texture image does not fit into memory
    - Process texture in tiles in order

- Texture scanning:
  - for v
    - for u
      - compute \(x(u,v)\) and \(y(u,v)\)
      - copy \(TEX[u,v]\) to \(SCR[x,y]\)
Surface Parameterization

• To apply textures we need 2D coordinates on surfaces
  → Parameterization

• Some objects have a natural parameterization
  – Sphere: spherical coordinates \((\varphi, \theta) = (2\pi u, \pi v)\)
  – Cylinder: cylindrical coordinates \((\varphi, h) = (2 \pi u, H v)\)
  – Parametric surfaces (such as B-spline or Bezier surfaces → later)

• Parameterization is less obvious for
  – Polygons, implicit surfaces, teapots…
Triangle Parameterization

- **Triangle is a planar object**
  - Has implicit parameterization (e.g. barycentric coordinates)
  - But we need more control: placement of triangle in texture space
- **Assign texture coordinates** \((u,v)\) to each vertex \((x_o,y_o,z_o)\)
- **Apply viewing projection** \((x_o,y_o,z_o) \rightarrow (x,y)\) (details later)
- **Yields full texture transformation** (warping) \((u,v) \rightarrow (x,y)\)

\[
\begin{align*}
x &= \frac{au + bv + c}{gu + hv + i} \\
y &= \frac{du + ev + f}{gu + hv + i}
\end{align*}
\]

- In homogeneous coordinates (by embedding \((u,v)\) as \((u,v,1)\))

\[
\begin{bmatrix}
x' \\
y' \\
w'
\end{bmatrix} =
\begin{bmatrix}
a & b & c \\
d & e & f \\
g & h & i
\end{bmatrix}
\begin{bmatrix}
u' \\
v' \\
w
\end{bmatrix};
(x, y) = \left(\frac{x'}{w'}, \frac{y'}{w'}\right), (u, v) = \left(\frac{u'}{q}, \frac{v'}{q}\right)
\]

- Transformation coefficients determined by 3 pairs \((u,v)\rightarrow(x,y)\)
  - Three linear equations
  - Invertible iff neither set of points is collinear
Triangle Parameterization (2)

- **Given**

  \[
  \begin{bmatrix}
  x' \\
y' \\
w'
  \end{bmatrix} =
  \begin{bmatrix}
  a & b & c \\
d & e & f \\
g & h & i
  \end{bmatrix}
  \begin{bmatrix}
  u' \\
v' \\
q
  \end{bmatrix}
  \]

- **The inverse transform** \((x,y) \rightarrow (u,v)\) is

  \[
  \begin{bmatrix}
  u' \\
v' \\
q
  \end{bmatrix} =
  \begin{bmatrix}
  ei - fh & ch - bi & bf - ce \\
f g - di & ai - cg & cd - af \\
dh - eg & bg - ah & ae - bd
  \end{bmatrix}
  \begin{bmatrix}
  x' \\
y' \\
w
  \end{bmatrix}
  \]

- **Coefficients must be calculated for each triangle**
  - Rasterization
    - Incremental bilinear update of \((u',v',q)\) in screen space
    - Using the partial derivatives of the linear function (i.e. constants)
  - Ray tracing
    - Evaluated at every intersection

- **Often derivatives are needed as well**
  - Explicitly given in matrix
Cylinder Parameterization

- Transformation from texture space to the cylinder parametric representation can be written as:

\[(\theta, h) = (2\pi u, vH)\]

- where \( H \) is the height of the cylinder.
- The surface coordinates in the Cartesian reference frame can be uniquely expressed as:

\[
\begin{align*}
    x_o &= r \cos \theta \\
    y_o &= r \sin \theta \\
    z_o &= h
\end{align*}
\]
Two-Stage Mapping

- Inverse mapping for arbitrary 3D surfaces too complex
- Approximation technique is used:
  - Mapping from 2D texture space to a simple 3D intermediate surface (S mapping)
    - Should be a reasonable approximation of the destination surface
    - E.g.: plane, cylinder, sphere, cube, ...
  - Mapping from the intermediate surface to the destination object surface (O mapping)
O-Mapping

- **Determine point on intermediate surface through**
  - Reflected view ray
    - Reflection or environment mapping
  - Normal mapping
  - Line through object centroid
  - Shrink-wrapping
    - Forward mapping
    - Normal mapping from intermediate surface
Two-Stage Mapping: Problems

- May introduce undesired texture distortions if the intermediate surface differs too much from the destination surface
- Still often used in practice because of its simplicity

Surface concavities can cause the texture pattern to reverse if the object normal mapping is used.
Two-Stage Mapping: Example

• Different intermediate surfaces

• Plane
  – Strong distortion where object surface normal $\perp$ to plane normal

• Cylinder
  – Reasonably uniform mapping (symmetry !)

• Sphere
  – Problems with concave regions
Projective Textures

- Project texture onto object surfaces
  - Slide projector

- Parallel or perspective projection

- Use photographs as textures

- Multiple images
  - View-dependent texturing (advanced topic)

- Perspective mapping
Projective Texturing: Examples
Reflection Mapping

- Also called “environment mapping”
- Reflection map parameterization
  - Intermediate surface in 2-stage mapping
  - Often cube, sphere, or double paraboloid
- Assumption: Distant illumination
  - Parallax-free illumination
  - No self-reflections, distortion of near objects
- Option: Separate map per object
  - Often necessary to be reasonably accurate
  - Reflections of other objects
  - Maps must be recomputed after changes
- Mirror reflections
  - Surface curvature: beam tracing
  - Map filtering
Reflection Map Acquisition

- Generating spherical maps (original 1982/83)
  - I.e. photo of a reflecting sphere (gazing ball)
Reflection Map Rendering

- Spherical parameterization
- O-mapping using reflected view ray intersection
Reflection Map Parameterization

• **Spherical mapping**
  – Single image
  – Bad utilization of the image area
  – Bad scanning on the edge
  – Artifacts, if map and image do not have the same viewpoint

• **Double parabolic mapping**
  – Yields spherical parameterization
  – Subdivide in 2 images (front-facing and back-facing sides)
  – Less bias near the periphery
  – Arbitrarily reusable
  – Supported by OpenGL extensions
Reflection Map Parameterization

- **Cubical environment map, cube map, box map**
  - Enclose object in cube
  - Images on faces are easy to compute
  - Poorer filtering at edges
  - Support in OpenGL
Reflection Mapping Example

Terminator II motion picture
Reflection Mapping Example II

- Reflection mapping with Phong reflection
  - Two maps: diffuse & specular
  - Diffuse: index by surface normal
  - Specular: indexed by reflected view vector
Ray Tracing vs. Reflection Map

• Differences?
Recursive Ray Tracing

- How to fake it with reflection mapping?

Figure 18.11
A recursive depth demonstration. The trace terminates at depth 2, 3, 4 and 5 (room image) respectively. ‘Unassigned’ pixels are colored grey. Bad aliasing as a function of recursive depth (the light cable) is apparent.
Light Maps

- **Light maps (e.g. in Quake)**
  - Pre-calculated illumination (local irradiance)
    - Often very low resolution: smoothly varying
  - Multiplication of irradiance with base texture
    - Diffuse reflectance only
  - Provides surface radiosity
    - View-independent out-going radiance
  - Animated light maps
    - Animated shadows, moving light spots, etc…

\[
B(x) = \rho(x) E(x) = \pi L_o(x)
\]

Reflectance \[\times\] Irradiance = Radiosity

Representing radiosity in a mesh or texture
Bump Mapping

• **Modulation of the normal vector**
  – Surface normals changed only
    • Influences shading only
    • No self-shadowing, contour is **not** altered
Bump Mapping

- **Original surface**: $O(u, v)$
  - Surface normals are known
- **Bump map**: $B(u, v) \in R$
  - Surface is offset in normal direction according to bump map intensity
  - New normal directions $N'(u, v)$ are calculated based on virtually displaced surface $O'(u, v)$
  - Original surface is rendered with new normals $N'(u, v)$
Bump Mapping

\[ O'(u, v) = O(u, v) + B(u, v) \frac{N}{|N|} \]

- Normal is cross-product of derivatives:

\[ O'_u = O_u + B_u \frac{N}{|N|} + B \left( \frac{N}{|N|} \right)_u \]
\[ O'_v = O_v + B_v \frac{N}{|N|} + B \left( \frac{N}{|N|} \right)_v \]

- If \( B \) is small (i.e. the bump map displacement function is small compared to its spatial extent) the last term in each equation can be ignored

\[ N'(u, v) = O_u \times O_v + B_u \left( \frac{N}{|N|} \times O_v \right) \]
\[ + B_v \left( O_u \times \frac{N}{|N|} \right) + B_u B_v \left( \frac{N \times N}{|N|^2} \right) \]

- The first term is the normal to the surface and the last is zero, giving:

\[ D = B_u (N \times O_v) - B_v (N \times O_u) \]
\[ N' = N + D \]
Texture Examples

• Complex optical effects
  – Combination of multiple texture effects

RenderMan Companion
Billboards

- **Single textured polygons**
  - Often with opacity texture
  - Rotates, always facing viewer
  - Used for rendering distant objects
  - Best results if approximately radially or spherically symmetric

- **Multiple textured polygons**
  - Azimuthal orientation: different view-points
  - Complex distribution: trunk, branches, …
3-D Textures

- “Carving object shape out of material block”
Texture Examples

- Solid 3D textures (wood, marble)
- Bump map (middle)
Part II

Procedural Methods
Texture Maps | Procedural Textures

- **Texture maps:** paintings, photos, videos, simulation...
  - Simple acquisition
  - Illumination “frozen” during acquisition
  - Limited resolution, aliasing
  - High memory requirements
  - Mapping issues

- **Procedural textures**
  - Non-trivial programming
  - Flexibility & parametric control
  - Unlimited resolution
  - Anti-aliasing possible
  - Low memory requirements
  - Low-cost visual complexity
  - Can adapt to arbitrary geometry
Procedural Textures

- Function of some shading parameter
  - E.g. world space, texture coordinates, ...

- Texturing: evaluation of function on object surface
  - Ray tracing: at intersection point with surface
  - Must be able to evaluate at random position efficiently

- Observation: textures of natural objects
  - Similarity between patches at different locations
    - Repetitiveness, coherence (e.g. skin of a tiger or zebra)
  - Similarity on different resolution scales
    - Self-similarity
  - But never completely identical
    - Additional disturbances, turbulence, noise

- Goal: generic procedural texture function
  - Mimics statistical properties of natural textures
  - Purely empirical approach
    - Looks convincing, but has nothing to do with material’s physics
Texture Examples

• Translational similarity

• Similarity on different scales

Romanesco broccoli [Wikipedia]
3D / Solid Noise: Perlin Noise

- **Noise(x,y,z)**
  - Statistical invariance under rotation
  - Statistical invariance under translation
  - Roughly one specific frequency

- **Integer lattice (i,j,k)**
  - Fixed fundamental frequency of ~1 Hz over lattice
  - Don’t store all values – use a hash function to randomize and look up from a fixed-size table
  - **Value noise**: Random value at lattice
  - **Gradient noise**: Random gradient vector at lattice point Q: G(Q)
    - Value at point P: G·(P-Q)
  - Tri-linear interpolation or cubic interpolation
    - Hermite spline → later

- **Unlimited domain due to lattice and hashing**
- **Also see**
  - http://www.noisemachine.com/talk1/
Noise vs. Noise

• **Gradient noise better than value noise**
  - Less regularity artifacts
  - More high frequencies in noise spectrum
  - Even tri-linear interpolation produces good results

• **Comparison between random values and Perlin noise**

Random values at each pixel

Gradient noise
Turbulence Function

- **Noise function**
  - Single spike in frequency spectrum

- **Natural textures**
  - Decreasing power spectrum towards high frequencies

- **Turbulence from noise**
  - $Turbulence(x) = \sum_{i=0}^{k} |a_i \ast noise(f_i x)|$
    - Frequency: $f_i = 2^i$
    - Amplitude: $a_i = 1 / p^i$
    - Persistence: $p$ typically $p=2$
  - Summation truncation
    - 1st term: noise(x)
    - 2nd term: noise(2x)/2
    - ...
      - Until period $(1/f_k) < 2$ pixel-size (band limit)
  - Power spectrum: $a_i = 1 / f_i$
  - Brownian motion: $a_i = 1 / f_i^2$
Synthesis of Turbulence (1D)
Synthesis of Turbulence (2D)
Example: Marble Texture Function

- Overall structure: alternating layers of white and colored marble
  - $f_{\text{marble}}(x,y,z) := \text{marble\_color}(\sin(x))$
  - $\text{marble\_color}$: transfer function (see lower left)

- Realistic appearance: simulated turbulence
  - $f_{\text{marble}}(x,y,z) := \text{marble\_color}(\sin(x+\text{turbulence}(x,y,z)))$

- Moving object: turbulence function also transformed
Further Procedural Texturing Applications

- **Bark**
  - Turbulated sawtooth function
  - Bump mapping

- **Clouds**
  - White blobs
  - Turbulated transparency along edge
  - Transparency mapping

- **Animation**
  - Vary procedural texture function’s parameters over time
Fractal Landscapes

- Procedural generation of geometry
- Complex geometry at virtually no memory cost
  - Can be difficult to ray trace !!
Fractal Landscapes

• **Coarse triangle mesh approximation**

• **1:4 triangle subdivision**
  – Vertex insertion at edge-midpoints

• **New vertex perturbation**
  – Random displacement along normal
  – Scale of perturbation depends on subdivision level
    • Decreasing power spectrum
    • Parameter models surface roughness

• **Recursive subdivision**
  – Level of detail (LOD) determined by # subdivisions

• **All done inside renderer!**
  – LOD generated locally when/where needed (bounding box test)
  – Minimal I/O cost (coarse mesh only)
Fractal Landscapes

- **Triangle subdivision**
  - Insert new vertices at edge midpoints
  - 1:4 triangle subdivision

- **Vertex displacement**
  - Along original triangle normal

Courtesy http://www.uni-paderborn.de/SFB376/projects/a2/zBufferMerging/
Fractal Landscape Generation

- Base mesh
- Repeated subdivision & vertex displacement
- Shading + Water surface + Fog + …

Courtesy http://www.uwp.edu/academic/computer.science/morris.csci/CS.320/Week.11/Ch11b.www/Ch11b.html
Fractal Landscape Ray Tracing

• Fractal terrain generated on-the-fly
• Problem: where is the ray-surface interaction?
  – Triangle mesh not a-priori known
• Solution: bounding boxes
  – Maximum possible bounding box around each triangle
  – Decreasing displacement amplitude: finite bounding box
• Algorithm
  – Intersect ray with bounding box
  – If hit, subdivide corresponding triangle
  – Compute bounding boxes of 4 new triangles
  – Test against 4 new bounding boxes
  – Iterate until termination criterion fulfilled (LOD / pixel size)