Procedural modeling and shadow mapping

Computer Graphics
CSE 167
Lecture 15
CSE 167: Computer graphics

• Procedural modeling
  – Height fields
  – Fractals
    • L-systems
  – Shape grammar

• Shadow mapping
3D modeling

• Definition: creating 3D objects/scenes and defining their appearance (texture etc.)

• Previous methods covered in this course
  – Triangle meshes
  – Curves
  – Surface patches

• Interactive modeling
  – Manually place vertices and/or control points
3D modeling

• Realistic scenes are extremely complex models containing millions or billions of primitives

• Data driven modeling
  – Scan model geometry from real world examples
    • Use 3D scanners or similar devices
    • Use photographs as textures

• Procedural modeling
  – Construct 3D models and/or textures algorithmically
Procedural modeling

• Wide variety of techniques for algorithmic model creation
• Used to create models too complex (or tedious) to build manually
  – Terrain, clouds
  – Plants, ecosystems
  – Buildings, cities
• Usually defined by a small set of data, or rules, that describes the overall properties of the model
  – Example: tree defined by branching properties and leaf shapes
• Model is constructed by an algorithm
  – Often includes randomness to add variety
  – For example, a single tree pattern can be used to model an entire forest

[Deussen et al.]
Procedural modeling

• Use some sort of randomness to make models more interesting, natural, less uniform
• Pseudorandom number generation algorithms
  – Produce a sequence of (apparently) random numbers based on some initial seed value
  – rand() generates random number between 0 and 1
• Pseudorandom sequences are repeatable, as one can always reset the sequence
  – srand(seed) initializes the random number generator
  – If the seed value is changed, a different sequence of numbers will be generated
  – Non-repeatable sequences can be generated with srand((unsigned)time(NULL));
Procedural modeling

• Height fields
• Fractals
  – L-systems
• Shape grammar
Height fields

• Landscapes are often constructed as height fields
• Regular grid on the ground plane
• Store a height value at each point
• Can store large terrain in memory
  – No need to store all grid coordinates: inherent connectivity
• Shape terrain by operations that modify the height at each grid point
• Height can be generated from grayscale values
  – Allows using image processing tools to create terrain height
Midpoint displacement algorithm

- Random midpoint displacement algorithm (one-dimensional)
- Similar for triangles, quadrilaterals

Start with single horizontal line segment.
Repeat for sufficiently large number of times
{
  Repeat over each line segment in scene
  {
    Find midpoint of line segment.
    Displace midpoint in Y by random amount.
    Reduce range for random numbers.
  }
}

Result: Mountain Range

Source: http://gameprogrammer.com/fractal.html#midpoint
Diamond square algorithm

- Begins with a 2D array of size $2^n + 1$
- Four corner points must be set to initial values
- Iteratively perform diamond and square steps
  - Diamond step
    - For each square in the array, set the midpoint of that square to be the average of the four corner points plus a random value.
  - Square step
    - For each diamond in the array, set the midpoint of that diamond to be the average of the four corner points plus a random value.
- At each iteration, reduce the magnitude of the random value
Fractals

- Definition: a curve or geometric figure, each part of which has the same statistical character as the whole
- Fractals are useful in modeling naturally occurring structures (e.g., eroded coastlines, snowflakes) in which similar patterns recur at progressively smaller scales, and in describing partly random or chaotic phenomena such as crystal growth, fluid turbulence, and galaxy formation

From Wikipedia
Mandelbrot set

- $Z$ and $C$ are complex numbers
- Initialize $Z$ with $0 + 0i$
- Pick any $C$ and iterate $Z$

$$Z_{\text{new}} = Z_{\text{old}}^2 + C$$

- If $C$ is diverging to infinity, then it is not part of the Mandelbrot set; otherwise, it is

Demo: [http://www.scale18.com/canvas2.html](http://www.scale18.com/canvas2.html)

Niccolò Tartaglia (1499–1577)
One of the first to use complex numbers
Fractals

• 3D Mandelbrot Zoom

https://www.youtube.com/watch?v=0clz6WLfWaY
Fractals

• Trip inside a 3D fractal (Kleinian) GPU realtime rendering
  
  https://www.youtube.com/watch?v=XIzScwydxOE
Fractal landscapes

• Add textures, material properties and use a nice rendering algorithm

• Example: Terragen Free (no cost version)
http://www.planetside.co.uk/terragen/

http://planetside.co.uk/terragen-image-gallery/
L-systems

• Developed by biologist Aristid Lindenmayer in 1968 to study growth patterns of algae

• Defined by grammar

\[ G = \{V, S, \omega, P\} \]

  – \( V \) alphabet, set of symbols that can be replaced (variables)
  – \( S \) set of symbols that remain fixed (constants)
  – \( \omega \) string of symbols defining initial state
  – \( P \) production rules

• Stochastic L-system

  – If there is more than one production rule for a symbol, then randomly choose one
Turtle interpretation for L-systems

- **Origin**: functional programming language Logo
  - Dialect of Lisp
  - Designed for education: drove a mechanical turtle as an output device

- **Turtle interpretation of strings**
  - State of turtle defined by \((x, y, \alpha)\) for position and heading
  - Turtle moves by step size \(d\) and angle increment \(\delta\)

- **Sample Grammar**
  - \(F\): move forward a step of length \(d\)
    - New turtle state: \((x', y', \alpha)\)
    - \(x' = x + d \cos \alpha\)
    - \(y' = y + d \sin \alpha\)
    - A line segment between points \((x, y)\) and \((x', y')\) is drawn
  - \(+\) Turn left by angle \(\delta\). Next state of turtle is \((x, y, \alpha + \delta)\)
    - Positive orientation of angles is counterclockwise
  - \(-\) Turn right by angle \(\delta\). Next state of turtle is \((x, y, \alpha - \delta)\)
Example: Sierpinski triangle

- **Variables**: A, B
  - Both mean draw forward
- **Constants**: +, -
  - Turn left by 60 degrees, right by 60 degrees
- **Start**: A
- **Rules**: (A → B-A-B), (B → A+B+A)
Example: fern

• Variables: $X$, $F$
  $X$ no drawing operation
  $F$ move forward

• Constants: $+$, $-$
  $+$ Turn left 25 degrees
  $-$ Turn right 25 degrees

• Start: $X$

• Rules:
  $(X \rightarrow F[-X][X]F[-X]+FX),(F \rightarrow FF)$

[Wikipedia]
[corresponds to saving the current values for position and angle (push), which are restored when the corresponding ] is executed (pop)
Examples

http://www.kevs3d.co.uk/dev/l systems/
Fractal trees

- Tutorial for recursive generation of trees in 3D: http://web.comhem.se/solgrop/3dtree.htm
  - Model trunk and branches as cylinders
  - Change color from brown to green at certain level of recursion

Dragon Curve Tree

Source: Allen Pike
Shape grammar

• Shape Rules
  – Defines how an existing shape can be transformed

• Generation Engine
  – Performs the transformations

• Working Area
  – Displays created geometry
Example: Coca-Cola bottle

Evolution of Coca-Cola bottles

Division of a Coca-Cola bottle
Example: Coca-Cola bottle

• Shape computation for two existing Coca-Cola bottles

Example: procedural buildings

• Demo fr-041: debris by Farbrausch, 2007
  https://www.youtube.com/watch?v=wqu_lpkOYBg&hd=1
  Single, 177 KB EXE file!
  http://www.farbrausch.de/
Shadow mapping
Shadows

• Give additional cues on scene lighting
Shadows

• Contact points
• Depth cues
Shadows

• Realism

Without self-shadowing

With self-shadowing
**Terminology**

**Umbra**: fully shadowed region

**Penumbra**: partially shadowed region
Hard and soft shadows

• Point and directional lights lead to hard shadows, no penumbra
• Area light sources lead to soft shadows, with penumbra
Hard and soft shadows

Hard shadow from point light source

Soft shadow from area light source
Shadows for interactive rendering

- Hard shadows only in this course
  - Soft shadows are difficult to compute in interactive graphics
- Two most popular techniques
  - Shadow mapping
  - Shadow volumes
- Many variations, subtleties
- Active research area
Shadow mapping

• A scene point is lit by the light source if visible from the light source

• Determine visibility from light source by placing a camera at the light source position and rendering the scene from there

Scene points are lit if visible from light source

Determine visibility from light source by placing camera at light source position
Two pass algorithm

• First Pass
  – Render scene by placing camera at light source position
  – Store depth image (shadow map)
Two pass algorithm

• Second Pass
  – Render scene from camera position
  – At each pixel, compare distance to light source with value in shadow map
    • If distance is larger, then pixel is in shadow
    • If distance is smaller or equal, then pixel is lit

Final image with shadows
Shadow map look-up

- Need to transform each point from object space to shadow map
- Shadow map texture coordinates are in $[0,1]^2$
- Transformation from object to shadow map coordinates

$$T = P_{\text{light}} V_{\text{light}} M$$

- $T$ is called texture matrix
- After perspective projection we have shadow map coordinates

Light space

Shadow map

Object space
Shadow map look-up

- Transform each vertex to normalized frustum of light
  \[
  \begin{bmatrix}
  s \\
  t \\
  r \\
  q
  \end{bmatrix} = \mathbf{T}
  \begin{bmatrix}
  x \\
  y \\
  z \\
  1
  \end{bmatrix}
  \]

- Pass \(s, t, r, q\) as texture coordinates to rasterizer
- Rasterizer interpolates \(s, t, r, q\) to each pixel
- Use projective texturing to look up shadow map
  - This means, the texturing unit automatically computes \((s/q, t/q, r/q, 1)\)
  - \((s/q, t/q)\) are shadow map coordinates in \([0,1]^2\)
  - \(r/q\) is depth in light space
- Shadow depth test: compare shadow map at \((s/q, t/q)\) to \(r/q\)
GLSL specifics

• In application
  – Store matrix $T$ in OpenGL texture matrix
  – Set using `glMatrixMode(GL_TEXTURE)`

• In vertex shader
  – Access texture matrix through predefined uniform `gl_TextureMatrix`

• In fragment shader
  – Declare shadow map as `sampler2DShadow`
  – Look up shadow map using projective texturing with `vec4 texture2DProj(sampler2D, vec4)`
Implementation specifics

• When doing a projective texture look up on a sampler2DShadow, the depth test is performed automatically
  – Return value is (1,1,1,1) if lit
  – Return value is (0,0,0,1) if shadowed

• Simply multiply result of shading with current light source with this value
Shadow mapping

• Demo

http://www.paulsprojects.net/opengl/shadowmap/shadowmap.html
Shadow mapping

• Tutorial

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
Shadow maps

• Issues
  – Sampling problems
  – Limited field of view (of shadow map)
  – Z-fighting
Sampling problems

- Shadow map pixel may project to many image pixels
  - Stair-stepping artifacts
Sampling problems

- **Solutions**
  - Increase resolution of shadow map
    - Not always sufficient
  - Split shadow map into several tiles
  - Modify projection for shadow map rendering
    - Light space perspective shadow maps (LiSPSM)
      [http://www.cg.tuwien.ac.at/research/vr/lispsm/](http://www.cg.tuwien.ac.at/research/vr/lispsm/)
  - Combination of splitting and LiSPSM
    - Basis for most commercial implementations
Limited field of view

- What if a scene point is outside the field of view of the shadow map?
Limited field of view

- What if a scene point is outside the field of view of the shadow map?
  - Use six shadow maps, arranged in a cube
- Requires a rendering pass for each shadow map
Z-fighting

- Depth values for points visible from light source are equal in both rendering passes
- Because of limited resolution, depth of pixel visible from light could be larger than shadow map value
- Need to add bias in first pass to make sure pixels are lit
Z-fighting

• Add bias when rendering shadow map
  – Move geometry away from light by small amount
• Finding correct amount of bias is tricky

Correct bias  Not enough bias  Too much bias
Z-fighting

Not enough bias

Too much bias

Just right
Shadow mapping

• Directional light sources
  – Shadow mapping can be done with orthographic projection in step one when creating the shadow map

• More information
  http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
Shadow mapping

• Resources
  – Overview (lots of links)
    http://www.realtimerendering.com/
  – Basic shadow maps
  – Avoiding sampling problems in shadow maps
    http://www.cg.tuwien.ac.at/research/vr/lispsm/
  – Faking soft shadows with shadow maps
    http://people.csail.mit.edu/ericchan/papers/smoothie/