CSE 167:
Introduction to Computer Graphics
Lecture #17: Procedural Modeling

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2011

Announcements

- Important dates:
  - Final project outline due November 23rd
    - Email to me at `jschulze@ucsd.edu`
  - Final project presentations: Friday December 2nd, 1-3pm, CSE room 1202
  - Final project web page due December 1st
  - Final Exam: December 9th, 3-6pm

Lecture Overview

- Shadow Mapping
  - Implementation
- Procedural Modeling
  - Concepts
  - Algorithms

Shadow Mapping With GLSL

First Pass

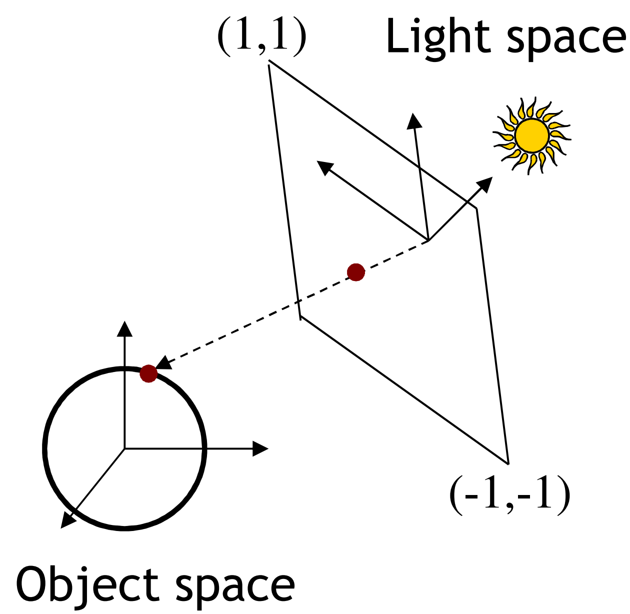
- Render scene by placing camera at light source position
- Compute light view (look at) matrix
  - Similar to computing camera matrix from look-at, up vector
  - Compute its inverse to get world-to-light transform
- Determine view frustum such that scene is completely enclosed
  - Use several view frusta/shadow maps if necessary

First Pass

- Each vertex point is transformed by

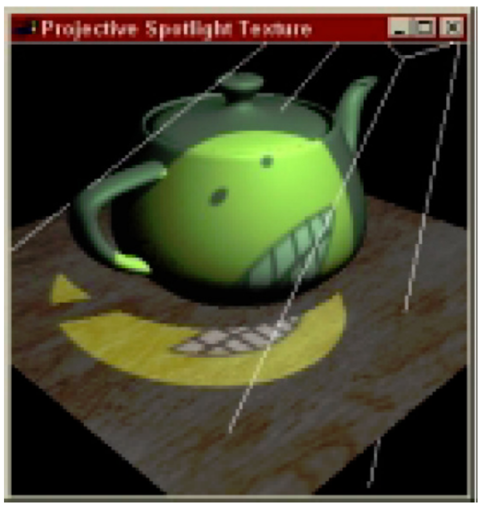
$$\mathbf{P}_{light}\mathbf{V}_{light}\mathbf{M}$$

  - Object-to-world (modeling) matrix $\mathbf{M}$
  - World-to-light space matrix $\mathbf{V}_{light}$
  - Light frustum (projection) matrix $\mathbf{P}_{light}$
- Remember: points within frustum are transformed to unit cube $[-1,1]^3$



First Pass

- Use glPolygonOffset to apply depth bias
- Store depth image in a texture
  - Use glCopyTexImage with internal format GL_DEPTH_COMPONENT



Final result
with shadows

Scene rendered
from light source

Depth map
from light source

Second Pass

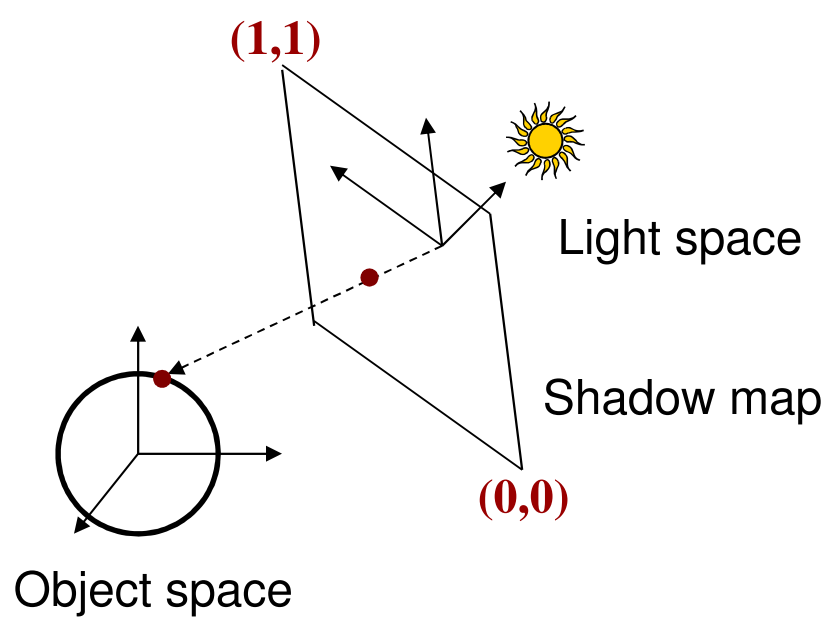
- Render scene from camera
- At each pixel, look up corresponding location in shadow map
- Compare depths with respect to light source

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

$$\mathbf{T} = \begin{bmatrix} 1/2 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{P}_{light} \mathbf{V}_{light} \mathbf{M}$$

- $\mathbf{T}$ is called texture matrix
- After perspective projection we have shadow map coordinates



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

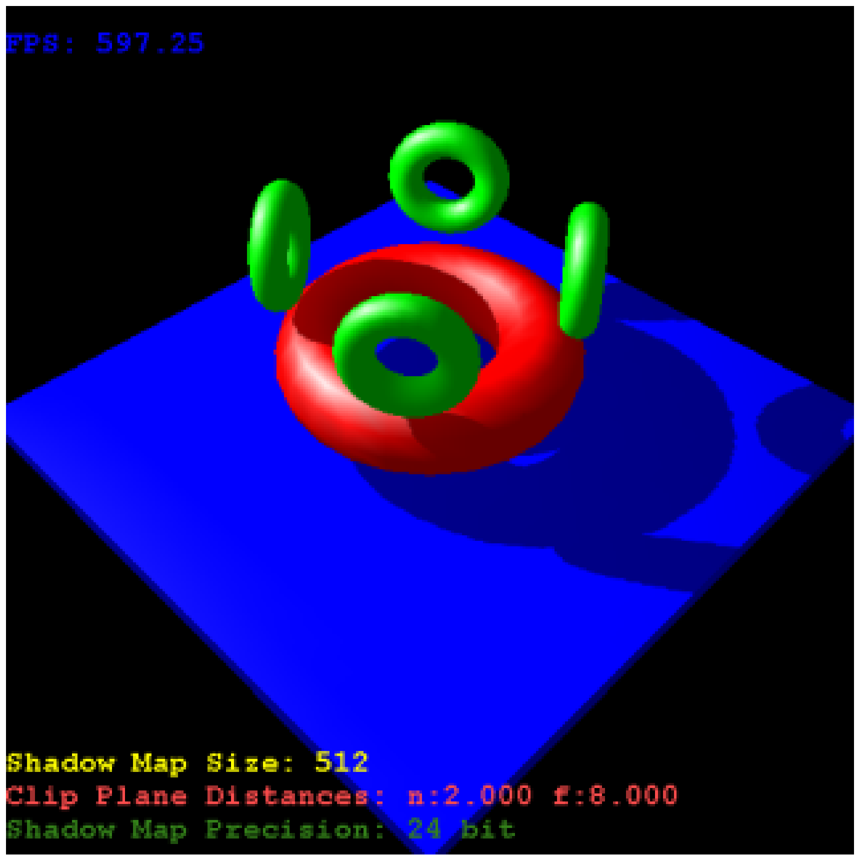
- Declare shadow map as sampler2DShadow
- Look up shadow map using projective texturing with `vec4 texture2DProj(sampler2D, vec4)`

Implementation Specifics

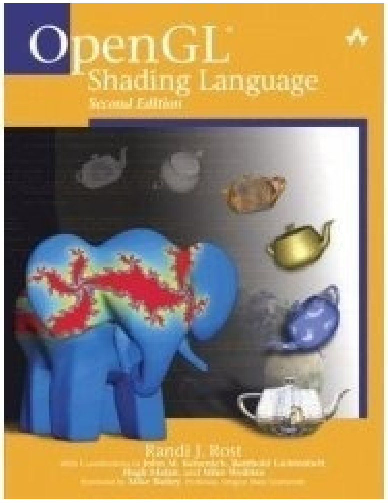
- When you do a projective texture look up on a `sampler2DShadow`, the depth test is performed automatically
  - Return value is (1,1,1,1) if lit
  - Return value is (0,0,0,1) if shadowed
- Simply multiply result of shading with current light source with this value

Demo

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



More on Shaders



- OpenGL shading language book
  - "Orange Book"
- Shader Libraries
  - GLSL:
    - http://www.geeks3d.com/geexlab/shader_library.php
  - HLSL:
    - NVidia shader library
    - http://developer.download.nvidia.com/shaderlibrary/webpages/shader_library.html

Lecture Overview

- Shadow Mapping
  - Implementation
- Procedural Modeling
  - Concepts
  - Algorithms

Modeling

- Creating 3D objects/scenes and defining their appearance (texture, etc.)
- So far we created
  - Triangle meshes
  - Bezier patches
- Interactive modeling
  - Place vertices, control points manually
- For realistic scenes, need extremely complex models containing millions or billions of primitives
- Modeling everything manually is extremely tedious

Alternatives

- Data-driven modeling
  - Scan model geometry from real world examples
  - Use laser scanners or similar devices
  - Use photographs as textures
  - Archives of 3D models
    - http://www-graphics.stanford.edu/data/3Dscanrep/ http://www.tsi.enst.fr/3dmodels/
    - Reader for PLY point file format: http://w3.impa.br/~diego/software/rply/

Photograph Rendering
[Levoy et al.]

- Procedural modeling
  - Construct 3D models and/or textures algorithmically

Procedural Modeling

[Deussen et al.]

- 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
  - Tree defined by branching properties and leaf shapes
- Model is constructed by an algorithm
  - Often includes randomness to add variety
  - E.g., a single tree pattern can be used to model an entire forest

Randomness

- 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
- Pseudorandom sequences are repeatable, as one can always reset the sequence
  - E.g., if a tree is built using pseudorandom numbers, then the entire tree can be rebuilt by resetting the seed value
  - If the seed value is changed, a different sequence of numbers will be generated, resulting in a (slightly) different tree

Recursion

- Repeatedly apply the same operation (set of operations) to an object
- Generate self-similar objects: fractals
  - Objects wich look similar when viewed at different scales
- For example, the shape of a coastline may appear as a jagged line on a map
  - As we zoom in, we see that there is more and more detail at finer scales
  - We always see a jagged line no matter how close we look at the coastline

Lecture Overview

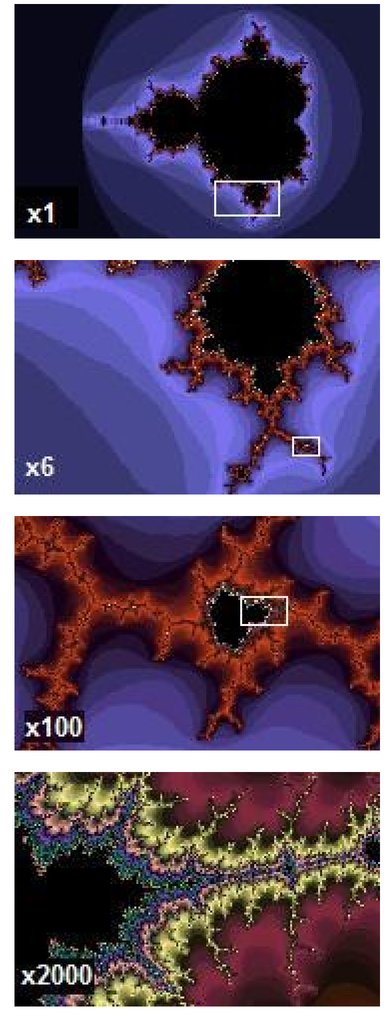
- Shadow Mapping
  - Implementation
- Procedural Modeling
  - Concepts
  - Algorithms

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
- Can generate height from grey scale values
  - Allows using image processing tools to create terrain height
  - → Extra credit in Homework Assignment #2

Fractals

- Fractal:
  Fragmented geometric shape which can be split into parts, each of which is (at least approximately) a smaller size copy of the whole
- Self-similarity
- Demo: Mandelbrot Set
  http://www.scale18.com/canvas2.html



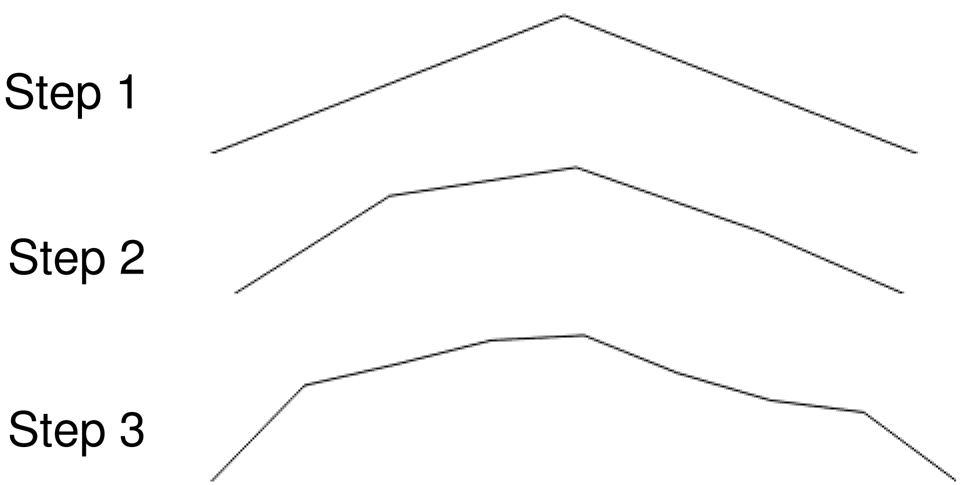
From Wikipedia

Fractal Landscapes

- Random midpoint displacement algorithm (one-dimensional)

```
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.
  }
}
```

- Similar for triangles, quadrilaterals



Result: Mountain Range

Source: http://gameprogrammer.com/fractal.html#midpoint

Fractal Landscapes

- Add textures, material properties; use nice rendering algorithm
- Example: Terragen Classic (free software) http://www.planetside.co.uk/terragen/

[http://www.planetside.co.uk/gallery/f/tg09]

L-Systems

- Developed by biologist Aristid Lindenmayer in 1968 to study growth patterns of algae
- Defined by grammar

$$\mathbf{G} = \{V, S, \omega, P\}$$

  - V = alphabet, set of symbols that can be replaced (variables)
  - S = set of symbols that remain fixed (constants)
  - ω = string of symbols defining initial state
  - P = production rules
- Stochastic L-system
  - If there is more than one production rule for a symbol, randomly choose one

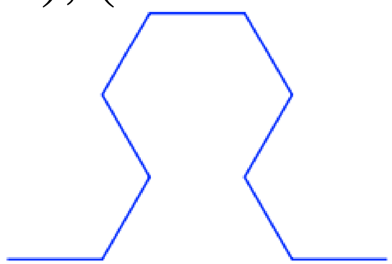
Turtle Interpretation for L-Systems

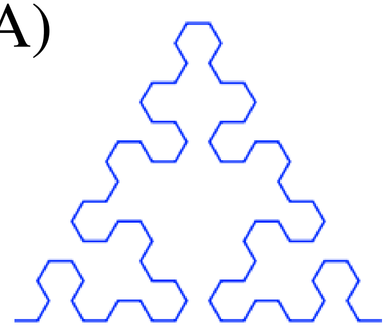
- Origin: functional programming language Logo
  - Dialect of Lisp
  - Designed for education: drove a mechanical turtle as an output device
- Turtle interpretation of strings
  - State of turtle defined by (x, y, α) *for* position and heading
  - Turtle moves by step size d and angle increment δ
- Sample Grammar
  - F: move forward a step of length d
    New turtle state: (x', y', α)
    $x' = x + d \cos \alpha$
    $y' = y + d \sin \alpha$
    A line segment between points (x, y) and (x', y') is drawn.
  - +: Turn left by angle δ. Next state of turtle is $(x, y, \alpha+\delta)$
    Positive orientation of angles is counterclockwise.
  - –: Turn right by angle δ. Next state of turtle is $(x, y, \alpha-\delta)$

Example: Sierpinski Triangle

- Variables: A, B
  - Draw forward
- Constants: + , -
  - Turn left, right by 60 degrees
- Start: A
- Rules: (A→B-A-B), (B→A+B+A)

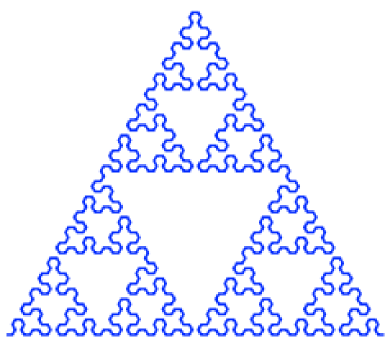
2 iterations

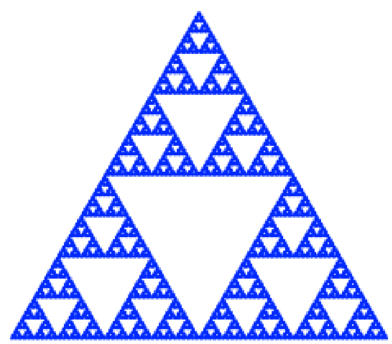




4 iterations

6 iterations





9 iterations

Example: Fern

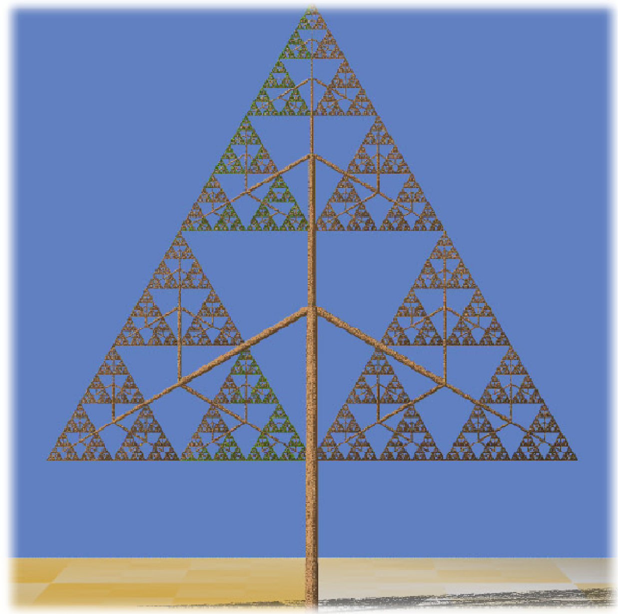
- Variables: X, F
  - X: no drawing operation
  - F: move forward
- Constants: +, –
  - Turn left, right
- Start: X
- Rules:
  (X → F-[[X]+X]+F[+FX]-X),(F → FF)

[Wikipedia]

Fractal Trees

- 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



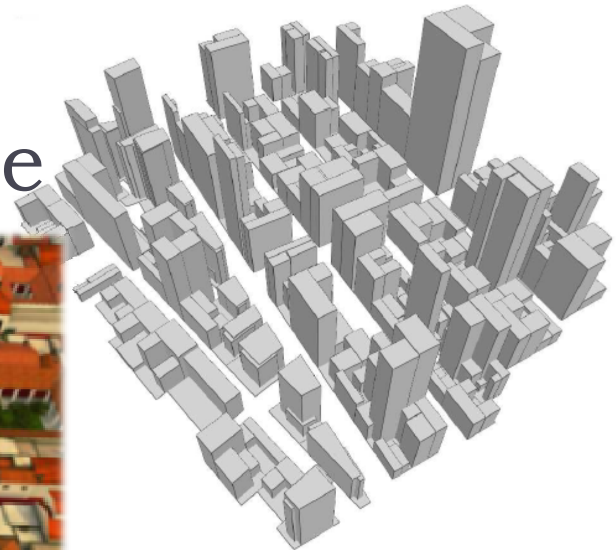
Sierpinski Tree

Algorithmic Beauty of Plants

- Book "The Algorithmic Beauty of Plants" by Przemyslaw Prusinkiewicz and Aristid Lindenmayer, 2004
- On-Line at: http://algorithmicbotany.org/papers/#abop

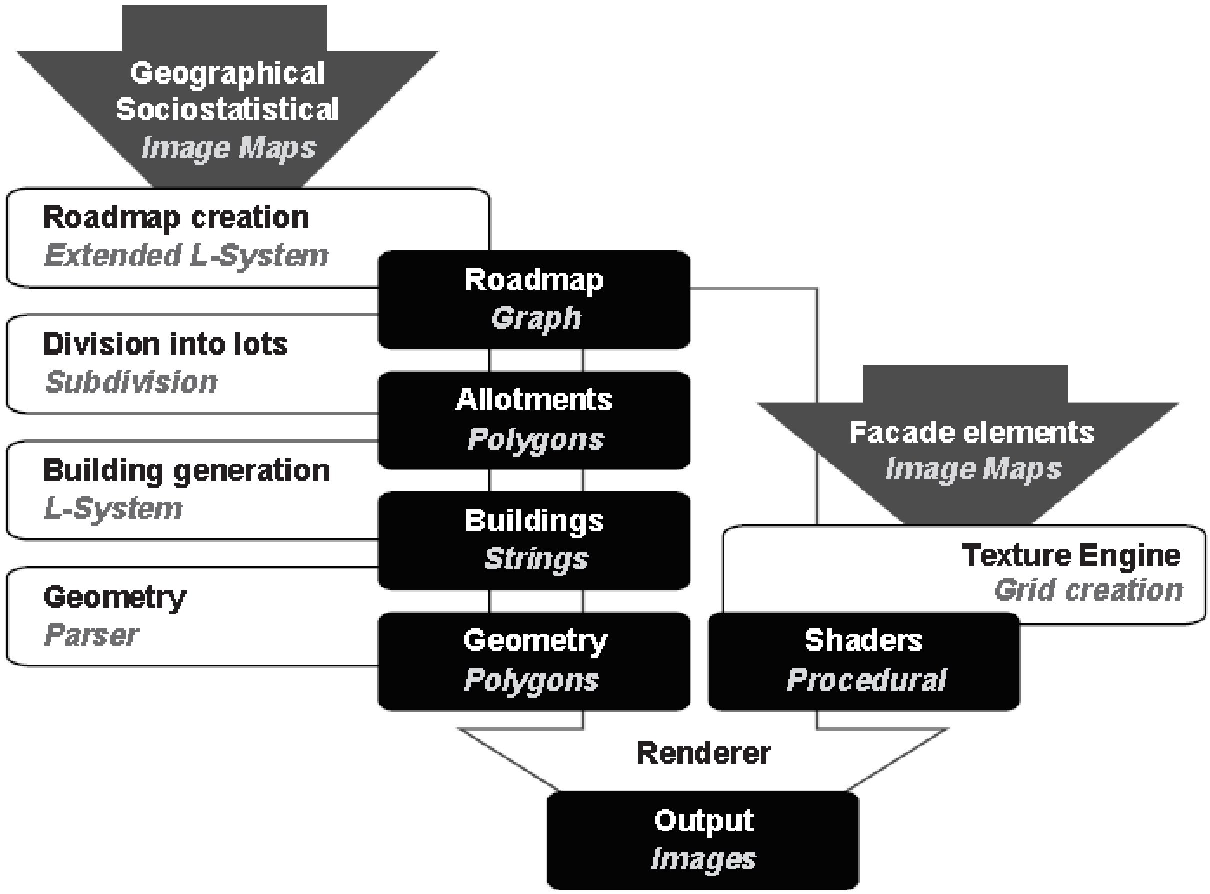
[Prusinkiewicz, http://algorithmicbotany.org/papers/positional.sig2001.pdf]

Buildings, Cities: CityEngine



http://www.esri.com/software/cityengine/

CityEngine: Pipeline



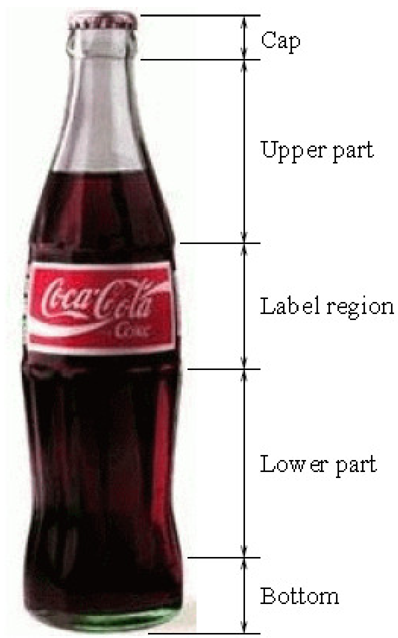
Parish, Mueller: “Procedural Modeling of Cities”, ACM Siggraph 2001

Shape Grammar

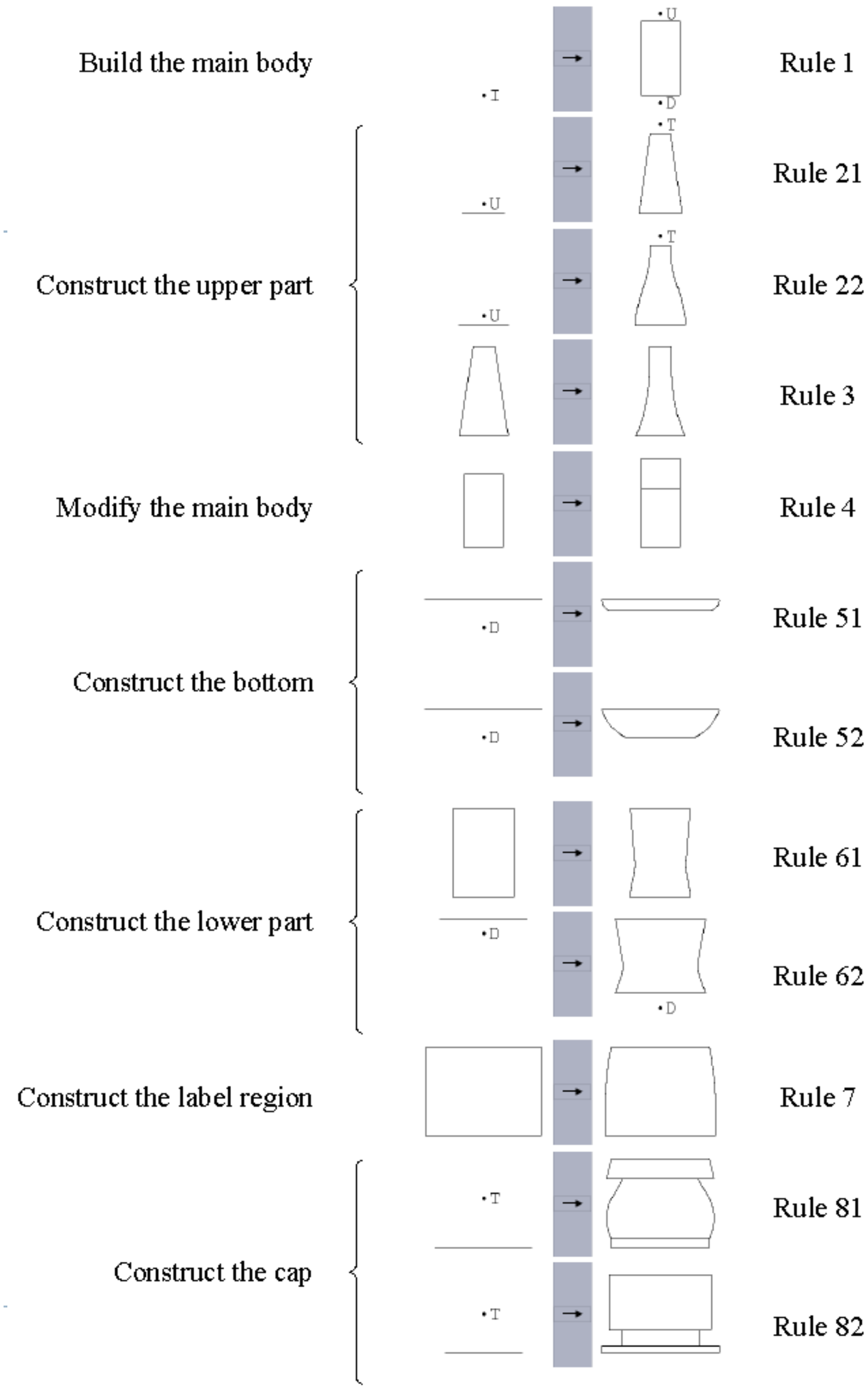
- Shape Rules
  - Defines how an existing shape can be transformed
- Generation Engine
  - Performs the transformations
- Working Area
  - Displays created geometry

Example: Coca-Cola Bottle

Evolution of Coca-Cola bottles

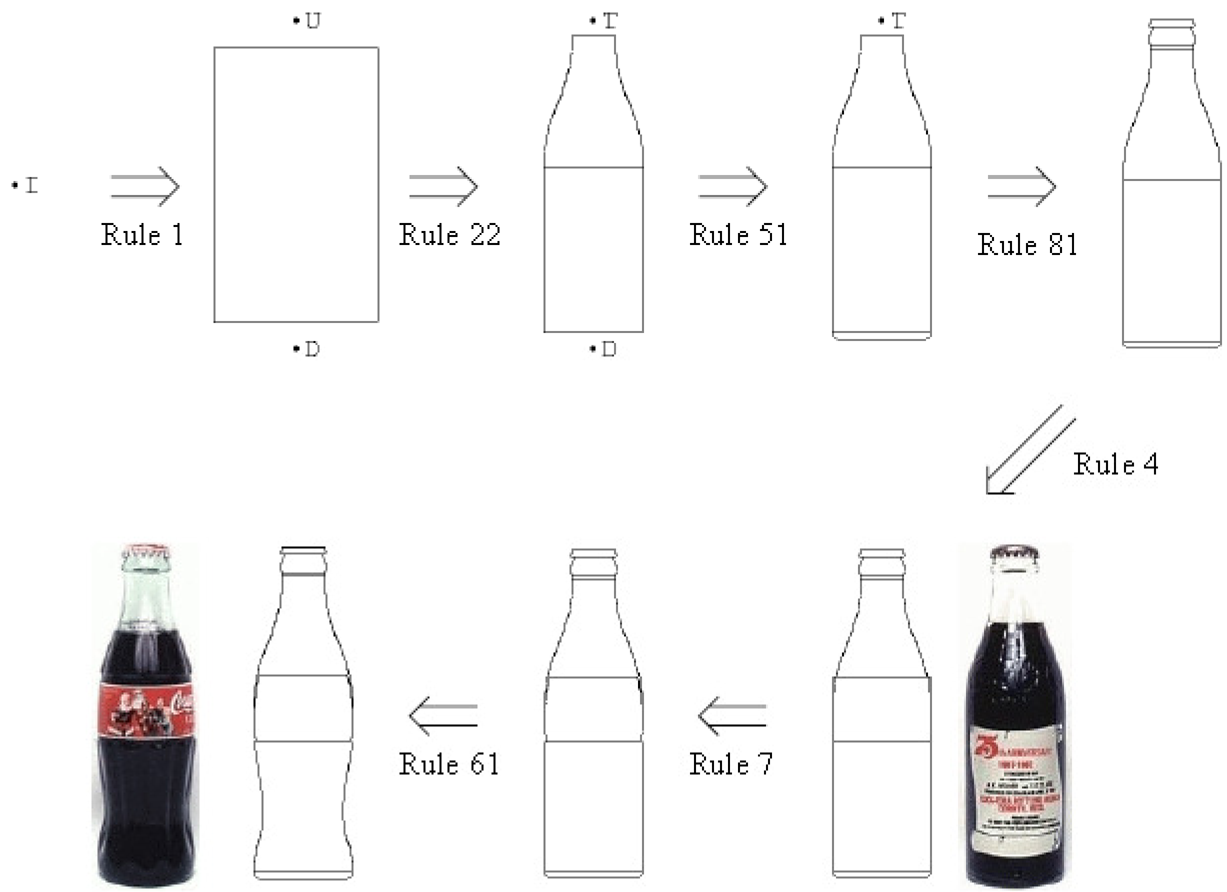


Division of a Coca-Cola bottle



Shape Computation Example

- Shape computation for two existing Coca-Cola bottles



Source: Chau et al.: “Evaluation of a 3D Shape Grammar Implementation”, *Design Computing and Cognition'04,* pp. 357-376

Demonstration: Procedural Buildings

- [Demo fr-041: debris by Farbrausch, 2007]()
- Single, 177 KB EXE file!
- http://www.farbrausch.de/

Next Lecture

- Volume Rendering