Computer Graphics
MTAT.03.015
Raimond Tunnel
mtllib triangle.mtl
o Plane
v 1.007839 0.000000 -1.000000
v 1.000000 0.000000 0.978599
v -1.000000 0.000000 -0.588960
usemtl None
s off
f 3 2 1
Procedural Generation

• Generating objects algorithmically

```cpp
for(y = 0; y <= heightSegments; y++) {
    for(x = 0; x <= widthSegments; x++) {
        u = (float)x / widthSegments;
        v = (float)y / heightSegments;

        glm::vec3 vertex = glm::vec3(
            -radius * glm::cos(phiStart + u * phiLength) * glm::sin(thetaStart + v * thetaLength),
            radius * glm::cos(thetaStart + v * thetaLength),
            radius * glm::sin(phiStart + u * phiLength) * glm::sin(thetaStart + v * thetaLength)
        );

        vertices.push_back(vertex);
        normals.push_back(glm::normalize(vertex));
        colors.push_back(color);
    }
}
```
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
  - Material (texture)
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
  - Material (texture)
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
  - Material (texture)
  - Effects (particles)

Custom B. Chopper solution by Siim Raudsepp
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
  - Material (texture)
  - Effects (particles)
  - Animation

Inverse kinematics
Procedural Generation

• Generating objects algorithmically
  • Mesh (geometry)
  • Material (texture)
  • Effects (particles)
  • Animation
  • Worlds
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
  - Material (texture)
  - Effects (particles)
  - Animation
  - Worlds

Infinite Procedural Infrastructured World Generation (MSc thesis) by Andreas Sepp
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
  - Material (texture)
  - Effects (particles)
  - Animation
  - Worlds
  - Characters, weapons, space ships, ...
Procedural Generation

- Generating objects algorithmically
  - Mesh (geometry)
  - Material (texture)
  - Effects (particles)
  - Animation
  - Worlds
  - Characters, weapons, space ships, ...

- More content, less repetitive work for artists
Tree

• Let's try to generate a tree branch structure.
Let's try to generate a tree branch structure.

We start with a trunk.
Tree

- From the trunk, make two branches to both sides.
- We also continue on the forward path.
Tree

- We repeat the process for the new segments.
Tree

- We repeat the process for the new segments.
Tree

- Decrease the length of the segments each time.
Tree

- Repeat again the same process.
Tree

- Introduce randomness.
Tree

- What if we want to store the generated structure?
Tree

- What if we want to store the generated structure?
- For example, this smaller tree:
Tree

• What if we want to store the generated structure?
• For example, this smaller tree:
• We should specify the structure and the parameters (length, angle).
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$. 
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.

Nonterminals can be changed by production rules.
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.

Nonterminals can be changed by production rules.

They do not „terminate“ the derivation.
Formal Grammar (Chomsky)

• Formal grammar consists of:
  • Set of nonterminal symbols $\mathcal{N}$.
  • Set of terminal symbols $\Sigma$. 

Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.

Terminals cannot be changed by production rules.
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.

They do “terminate“ the derivation.
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.
  - Set of production rules.
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.
  - Set of production rules.

Rules tell you what nonterminals can be replaced with other nonterminals or terminals.
Formal Grammar (Chomsky)

• Formal grammar consists of:
  • Set of nonterminal symbols $N$.
  • Set of terminal symbols $\Sigma$.
  • Set of production rules.
  • Starting axiom.
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.
  - Set of production rules.
  - Starting axiom.

The initial „word“ of symbols / system state.
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.
  - Set of production rules.
  - Starting axiom.

- Example:

$$N = \{ A \}$$
Formal Grammar (Chomsky)

• Formal grammar consists of:
  • Set of nonterminal symbols $N$.
  • Set of terminal symbols $\Sigma$.
  • Set of production rules.
  • Starting axiom.

• Example:

\[ N = \{ A \} \]
\[ \Sigma = \{ a \} \]
Formal Grammar (Chomsky)

• Formal grammar consists of:
  • Set of nonterminal symbols $N$.
  • Set of terminal symbols $\Sigma$.
  • Set of production rules.
  • Starting axiom.

• Example:

$N = \{ A \}$  \hspace{1cm} $R = \{ A \rightarrow AA \}$  \hspace{1cm} $\Sigma = \{ a \}$
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.
  - Set of production rules.
  - Starting axiom.

- Example:

$$N = \{ A \} \quad R = \begin{cases} A \rightarrow AA \\ A \rightarrow a \end{cases}$$

$$\Sigma = \{ a \}$$

$Axiom = A$
Formal Grammar (Chomsky)

• Formal grammar consists of:
  • Set of nonterminal symbols \( N \).
  • Set of terminal symbols \( \Sigma \).
  • Set of production rules.
  • Starting axiom.

• Example:

\[
\begin{align*}
N & = \{ A \} \\
\Sigma & = \{ a \} \\
R & = \begin{cases} 
A \rightarrow AA \\
A \rightarrow a
\end{cases}
\end{align*}
\]

Generation:

\[A \rightarrow a\]

Axiom = A
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.
  - Set of production rules.
  - Starting axiom.

- Example:

\[
N = \{ A \} \\
R = \left\{ \begin{array}{l}
    A \rightarrow AA \\
    A \rightarrow a
\end{array} \right\} \\
\Sigma = \{ a \} \\
Axiom = A
\]

Generation:

\[
A \rightarrow a \\
A \rightarrow AA \rightarrow aA \rightarrow aa
\]
Formal Grammar (Chomsky)

• Formal grammar consists of:
  • Set of nonterminal symbols $N$.
  • Set of terminal symbols $\Sigma$.
  • Set of production rules.
  • Starting axiom.

• Example:

$N = \{ A \}$  \hspace{1cm} $R = \begin{cases} A \rightarrow AA \\ A \rightarrow a \end{cases}$

$\Sigma = \{ a \}$  \hspace{1cm} Axiom = $A$

Generation:

$$A \rightarrow a$$

$$A \rightarrow AA \rightarrow aA \rightarrow aa$$

$$A \rightarrow AA \rightarrow AAA \rightarrow aAA \rightarrow aaA \rightarrow aaa$$
Formal Grammar (Chomsky)

- Formal grammar consists of:
  - Set of nonterminal symbols $N$.
  - Set of terminal symbols $\Sigma$.
  - Set of production rules.
  - Starting axiom.

- Example:

  $$N = \{ A \} \quad R = \begin{cases} A \rightarrow AA \\ A \rightarrow a \end{cases} \quad \Sigma = \{ a \} \quad \text{Axiom} = A$$

- Generation:

  $$A \rightarrow a$$
  $$A \rightarrow AA \rightarrow aA \rightarrow aa$$
  $$A \rightarrow AA \rightarrow AAA \rightarrow aAA \rightarrow aaA \rightarrow aaa$$
  $$\ldots$$
Formal Grammar (Chomsky)

- Used for:
Formal Grammar (Chomsky)

● Used for:
  ● Natural language processing
Formal Grammar (Chomsky)

• Used for:
  • Natural language processing
  • Program code processing (compiler, interpreter)
Formal Grammar (Chomsky)

- Used for:
  - Natural language processing
  - Program code processing

- Hierarchy of types
  - Type 0: Unrestricted \( N = \Sigma \)
Formal Grammar (Chomsky)

- Used for:
  - Natural language processing
  - Program code processing

- Hierarchy of types
  - Type 0: Unrestricted – $N = \Sigma$
  - Type 1: Context sensitive – non-terminal symbol on the left side, can be surrounded by a context
Formal Grammar (Chomsky)

- Used for:
  - Natural language processing
  - Program code processing

- Hierarchy of types
  - **Type 0: Unrestricted** – $N = \Sigma$
  - **Type 1: Context sensitive** – non-terminal symbol on the left side, can be surrounded by a context
  - **Type 2: Context free** – left side contains only a single non-terminal symbol
Formal Grammar (Chomsky)

- Used for:
  - Natural language processing
  - Program code processing

- Hierarchy of types
  - **Type 0: Unrestricted** – $N = \Sigma$
  - **Type 1: Context sensitive** – non-terminal symbol on the left side, can be surrounded by a context
  - **Type 2: Context free** – left side contains only a single non-terminal symbol
  - **Type 3: Regular** – right side is empty, single terminal, or single terminal follower by non-terminal
Lindenmayer System

- Variant of a formal grammar.
Lindenmayer System

- Variant of a formal grammar.
- Parallel rewriting system.
Lindenmayer System

- Variant of a formal grammar.
- Parallel rewriting system.  

Because of that, does not fall directly under Chomsky's hierarchy
Lindenmayer System

- Variant of a formal grammar.
- Parallel rewriting system.
- We will look at one, that is:
  - Bracketed system.

Because of that, does not fall directly under Chomsky's hierarchy
Lindenmayer System

• Variant of a formal grammar.

• Parallel rewriting system.

• We will look at one, that is:
  • Bracketed system.
  • Stochastic system.

Because of that, does not fall directly under Chomsky's hierarchy.
Lindenmayer System

- Variant of a formal grammar.
- Parallel rewriting system.
- We will look at one, that is:
  - Bracketed system.
  - Stochastic system.
  - Context free (0L-system).

Because of that, does not fall directly under Chomsky's hierarchy.
Lindenmayer System

• Variant of a formal grammar.
• Parallel rewriting system.
• We will look at one, that is:
  • Bracketed system.
  • Stochastic system.
  • Context free (0L-system).
  • Parametric system.
Lindenmayer System

- **Bracketed system** – use brackets to indicate branches.
Lindenmayer System

- **Bracketed system** – use brackets to indicate branches.
- Using following symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Segment</td>
</tr>
<tr>
<td>+</td>
<td>Rotate left 45°</td>
</tr>
<tr>
<td>-</td>
<td>Rotate right 45°</td>
</tr>
<tr>
<td>[</td>
<td>Start of a branch</td>
</tr>
<tr>
<td>]</td>
<td>End of a branch</td>
</tr>
</tbody>
</table>

Can we write our tree using those?
Lindenmayer System

- Parallel rewriting system – all the rules will be applied in parallel to rewrite the entire word.
Lindenmayer System

- Parallel rewriting system – all the rules will be applied in parallel to rewrite the entire word.

What would be the rules to create the following?

Axiom: F
Lindenmayer System

- Parallel rewriting system – all the rules will be applied in parallel to rewrite the entire word.

What would be the rules to create the following?

Axiom: F
1. iteration: F[+F][-F]F
Lindenmayer System

- Parallel rewriting system – all the rules will be applied in parallel to rewrite the entire word.

What would be the rules to create the following?

Axiom: F
1. iteration: F[+F][-F]F
2. iteration:
F[+F[+F][-F]F]
[-F[+F][-F]F]
F[+F][-F]F

45° 45°
Lindenmayer System

- Parallel rewriting system – all the rules will be applied in parallel to rewrite the entire word.

What would be the rules to create the following?

Axiom: F

1. iteration: F[+F][-F]F


This is a trick question.
Lindenmayer System

- **Parametric system** – we can specify parameters for some of the symbols.
Lindenmayer System

• **Parametric system** – we can specify parameters for some of the symbols.
  • The length, the angle etc
Lindenmayer System

- **Parametric system** – we can specify parameters for some of the symbols.
- The length, the angle etc

\[
\begin{align*}
F[+(45)F[+(45)F][-(-45)F]F] \\
F[+(45)F][--(45)F]F
\end{align*}
\]

Every + or - is followed by the angle of rotation.
Lindenmayer System

- We can generate angles with some variance.

\[ F[+(31.24)F][-(47.89)F]F \]
Lindenmayer System

- We can generate **angles** with some variance.
- We can also specify the **lengths** of the segments.

\[
F(1)[+ (31.24) F(0.75)] [-(47.89) F(0.75)] F(0.75)
\]
Lindenmayer System

- We can generate angles with some variance.
- We can also specify the lengths of the segments.

If the decrease of lengths is deterministic, we could consider it only, when drawing the tree...

\[ F(1)[+(31.24)F(0.75)][-(47.89)F(0.75)]F(0.75) \]
Lindenmayer System

- **Stochastic system** – we can have many rules, with the same left-hand side.

\[
\begin{align*}
A & \rightarrow F[+A]A \\
A & \rightarrow F[-A]A \\
A & \rightarrow F[+A][-A]
\end{align*}
\]
Lindenmayer System

- **Stochastic system** – we can have many rules, with the same left-hand side.

- Each rule has a **probability**.

\[
\begin{align*}
A & \rightarrow \frac{1}{3} F[+A]A \\
A & \rightarrow \frac{1}{3} F[-A]A \\
A & \rightarrow \frac{1}{3} F[+A][-A]
\end{align*}
\]
Lindenmayer System

- **Stochastic system** – we can have many rules, with the same left-hand side.
- Each rule has a **probability**.
- The **sum** of the probabilities of all the rules, with the same left-hand side, **has to be 1**.

\[
\begin{align*}
A & \rightarrow \frac{1}{3} F[+A]A \\
A & \rightarrow \frac{1}{3} F[-A]A \\
A & \rightarrow \frac{1}{3} F[+A][-A]
\end{align*}
\]
Lindenmayer System

- Rigorous way to specify a mechanism for a self-similar structure generation.
Lindenmayer System

- Rigorous way to specify a mechanism for a self-similar structure generation.
Lindenmayer System

- Rigorous way to specify a mechanism for a self-similar structure generation.
Lindenmayer System

- Rigorous way to specify a mechanism for a self-similar structure generation.

recursive

fractal?
Lindenmayer System

- Rigorous way to specify a mechanism for a self-similar structure generation.
- Lot of research and possibilities.
Lindenmayer System

- Rigorous way to specify a mechanism for a self-similar structure generation.
- Lot of research and possibilities.
  
  http://algorithmicbotany.org/papers/abop/abop.pdf
Lindenmayer System

- Rigorous way to specify a mechanism for a self-similar structure generation.
- Lot of research and possibilities.
- Try out 2D online:
  [http://www.kevs3d.co.uk/dev/lsystems/](http://www.kevs3d.co.uk/dev/lsystems/)
Lindenmayer System

• Rigorous way to specify a mechanism for a self-similar structure generation.

• Lot of research and possibilities.

• *The Algorithmic Beauty of Plants*,
  A. Lindenmayer, P. Prusinkiewicz.
  http://algorithmicbotany.org/papers/abop/abop.pdf

• Try out 2D online:
  http://www.kevs3d.co.uk/dev/lsystems/

• Questions?
Lindenmayer System

F -1.000000  →  H[+G][-G]
G -0.200000  →  H[&G]
G -0.200000  →  H[+G]
G -0.200000  →  H[+G][-G]
G -0.200000  →  H[&G][-G]
H -0.900000  →  H
H -0.050000  →  H[&F]
H -0.050000  →  H[+F]
Particle System

• Used for different effects
  • Fire, fluid, wind, smoke
  • Precipitation (rain, snow)
  • Groups of objects with behaviour (birds, NPC-s)

This you did in the Soft Particle Chopper.

Simulating the Collective Movement of Fish Schools by Erik Martin Vetemaa
Particle System

- Particles can have a transparency that varies over time.
Particle System

- Particles can have a transparency that varies over time.
- Particles can be generated from an object pool.
  - If a particle dies, return it to the object pool.
Particle System

- Particles can have a transparency that varies over time.
- Particles can be generated from an object pool.
  - If a particle dies, return it to the object pool.
- Particle can be 1 pixel in size, or have an image.
Particle System

- Particles can have a transparency that varies over time.
- Particles can be generated from an object pool.
  - If a particle dies, return it to the object pool.
- Particle can be 1 pixel in size, or have an image.
- Particle system has an emitter of particles.

Emitter can also be a line, a surface, a volume etc.
Boids Algorithm

• By Craig Reynolds in 1986
Boids Algorithm

- By Craig Reynolds in 1986
- Used to model flocking (*eg of birds*).
Boids Algorithm

- By Craig Reynolds in 1986
- Used to model flocking (e.g., birds)

Each particle follows the rules:

- **Cohesion** – Move towards the center of mass.
Boids Algorithm

- By Craig Reynolds in 1986
- Used to model flocking (e.g., birds)

Each particle follows the rules:

- **Cohesion** – Move towards the center of mass.
- **Separation** – Keep distance from other particles.
Boids Algorithm

- By Craig Reynolds in 1986
- Used to model flocking (e.g., birds)

Each particle follows the rules:

- **Cohesion** – Move towards the center of mass.
- **Separation** – Keep distance from other particles.
- **Alignment** – Follow the average direction.
Boids Algorithm

- By Craig Reynolds in 1986
- Used to model flocking (eg of birds)

- Each particle follows the rules:
  - **Cohesion** – Move towards the center of mass.
  - **Separation** – Keep distance from other particles.
  - **Alignment** – Follow the average direction.

- There can be other rules.
Particle Systems

• Blender has particle systems

• Example of scar generation via particles:
  https://www.youtube.com/watch?v=e3FpG3CF1fQ
Particle Systems

• The Particles System task in the GE module.
What was new for you today?

What more would you like to know?

Next time: Ray Casting, Ray Tracing, Space Partitioning, BVH