Game Programming

Bing-Yu Chen
National Taiwan University
Game Geometry

- Graph and Meshes
- Surface Properties
- Bounding Volumes
- Spatial Partitioning
- Level-of-Details
Standard Graph Definitions

\[ G = <V,E> \]
\[ V = \text{vertices} = \{A,B,C,D,E,F,G,H,I,J,K,L\} \]
\[ E = \text{edges} = \{(A,B),(B,C),(C,D),(D,E),(E,F),(F,G),\]
\[ (G,H),(H,A),(A,J),(A,G),(B,J),(K,F),\]
\[ (C,L),(C,I),(D,I),(D,F),(F,I),(G,K),\]
\[ (J,L),(J,K),(K,L),(L,I)\} \]

**Vertex degree (valence)** = number of edges incident on vertex
Ex. \( \text{deg}(J) = 4 \), \( \text{deg}(H) = 2 \)

**k-regular graph** = graph whose vertices all have degree \( k \)

**Face**: cycle of vertices/edges which cannot be shortened
\[ (D,E,F),(D,I,F),(L,I,F,K),(L,J,K),(K,F,G)\} \]
Meshes

**Mesh**: straight-line graph embedded in $\mathbb{R}^3$

**Boundary** edge: adjacent to exactly one face

**Regular** edge: adjacent to exactly two faces

**Singular** edge: adjacent to more than two faces

Corners $\subseteq V \times F$
Half-edges $\subseteq E \times F$

**Closed** Mesh: mesh with no boundary edges

**Manifold** Mesh: mesh with no singular edges
Orientability

**Orientation** of a face is clockwise or anticlockwise order in which its vertices and edges are lists.

This defines the direction of face **normal**.

Straight line graph is **orientable** if orientations of its faces can be chosen so that each edge is oriented in both directions.

**Oriented**

\[ F = \{ (L,J,B), (B,C,L), (L,C,I), (I,K,L), (L,K,J) \} \]

**Not Oriented**

\[ F = \{ (B,J,L), (B,C,L), (L,C,I), (L,I,K), (L,K,J) \} \]

Back Face Culling = Front Facing
Mesh Data Structures

- Uses of mesh data:
  - Rendering
  - Geometry queries
    - What are the vertices of face #3?
    - Are vertices i and j adjacent?
    - Which faces are adjacent face #7?
  - Geometry operations
    - Remove/add a vertex/face
    - Mesh simplification
    - Vertex split, edge collapse

- Storage of generic meshes
  - hard to implement efficiently

- Assume: **orientable, manifold** and **triangular**
Storing Mesh Data

- How “good” is a data structure?
  - Time to construct – preprocessing
  - Time to answer a query
  - Time to perform an operation
    - update the data structure
  - Space complexity
  - Redundancy
1. List of Faces

- List of vertices (coordinates)

- List of faces
  - triplets of pointers to face vertices \((c_1, c_2, c_3)\)

- Queries:
  - What are the vertices of face #3?
    - \(O(1)\) – checking the third triplet
  - Are vertices \(i\) and \(j\) adjacent?
    - A pass over all faces is necessary – NOT GOOD
1. List of Faces

Example

<table>
<thead>
<tr>
<th>vertex</th>
<th>coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>v₁</td>
<td>(x₁,y₁,z₁)</td>
</tr>
<tr>
<td>v₂</td>
<td>(x₂,y₂,z₂)</td>
</tr>
<tr>
<td>v₃</td>
<td>(x₃,y₃,z₃)</td>
</tr>
<tr>
<td>v₄</td>
<td>(x₄,y₄,z₄)</td>
</tr>
<tr>
<td>v₅</td>
<td>(x₅,y₅,z₅)</td>
</tr>
<tr>
<td>v₆</td>
<td>(x₆,y₆,z₆)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>face</th>
<th>vertices (ccw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td>(v₁,v₂,v₃)</td>
</tr>
<tr>
<td>f₂</td>
<td>(v₂,v₄,v₃)</td>
</tr>
<tr>
<td>f₃</td>
<td>(v₃,v₄,v₆)</td>
</tr>
<tr>
<td>f₄</td>
<td>(v₄,v₅,v₆)</td>
</tr>
</tbody>
</table>
1. List of Faces

- Pros:
  - Convenient and efficient (memory wise)
  - Can represent non-manifold meshes

- Cons:
  - Too simple – not enough information on relations between vertices and faces
OBJ File Format (simple ver.)

- **v**  \( x \ y \ z \)
- **vn**  \( i \ j \ k \)
- **f**  \( v1 // vn1 \ v2 // vn2 \ v3 // vn3 \)
2. Adjacency matrix

☐ View mesh as connected graph
☐ Given n vertices build nxn matrix of adjacency information
  ■ Entry (i,j) is TRUE value if vertices i and j are adjacent
☐ Geometric info
  ■ list of vertex coordinates
☐ Add faces
  ■ list of triplets of vertex indices \((v_1,v_2,v_3)\)
2. Adjacency matrix

Example

<table>
<thead>
<tr>
<th>vertex</th>
<th>coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>($x_1, y_1, z_1$)</td>
</tr>
<tr>
<td>$v_2$</td>
<td>($x_2, y_2, z_2$)</td>
</tr>
<tr>
<td>$v_3$</td>
<td>($x_3, y_3, z_3$)</td>
</tr>
<tr>
<td>$v_4$</td>
<td>($x_4, y_4, z_4$)</td>
</tr>
<tr>
<td>$v_5$</td>
<td>($x_5, y_5, z_5$)</td>
</tr>
<tr>
<td>$v_6$</td>
<td>($x_6, y_6, z_6$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>face</th>
<th>vertices (ccw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>($v_1, v_2, v_3$)</td>
</tr>
<tr>
<td>$f_2$</td>
<td>($v_2, v_4, v_3$)</td>
</tr>
<tr>
<td>$f_3$</td>
<td>($v_3, v_4, v_6$)</td>
</tr>
<tr>
<td>$f_4$</td>
<td>($v_4, v_5, v_6$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$v_2$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$v_3$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$v_4$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$v_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$v_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
2. Adjacency matrix

- Queries:
  - What are the vertices of face #3?
    - O(1) – checking the third triplet of faces
  - Are vertices i and j adjacent?
    - O(1) – checking adjacency matrix at location (i,j)
  - Which faces are adjacent of vertex j?
    - Full pass on all faces is necessary
2. Adjacency matrix

- **Pros:**
  - Information on vertices adjacency
  - Stores non-manifold meshes

- **Cons:**
  - Connects faces to their vertices, BUT NO connection between vertex and its face
3. DCEL
(Doubly-Connected Edge List)

- Record for each face, edge and vertex
  - Geometric information
  - Topological information
  - Attribute information

- aka Half-Edge Structure
3. DCEL

- Vertex record:
  - Coordinates
  - Pointer to one half-edge that has v as its origin

- Face record:
  - Pointer to one half-edge on its boundary

- Half-edge record:
  - Pointer to its origin, \text{origin}(e)
  - Pointer to its twin half-edge, \text{twin}(e)
  - Pointer to the face it bounds, \text{IncidentFace}(e)
    - face lies to left of e when traversed from origin to destination
  - Next and previous edge on boundary of \text{IncidentFace}(e), \text{next}(e) and \text{prev}(e)
3. DCEL

- Operations supported:
  - Walk around boundary of given face
  - Visit all edges incident to vertex v

- Queries:
  - Most queries are $O(1)$
3. DCEL

Example

<table>
<thead>
<tr>
<th>vertex</th>
<th>coordinate</th>
<th>IncidentEdge</th>
</tr>
</thead>
<tbody>
<tr>
<td>v₁</td>
<td>(x₁, y₁, z₁)</td>
<td>e₂,₁</td>
</tr>
<tr>
<td>v₂</td>
<td>(x₂, y₂, z₂)</td>
<td>e₁,₁</td>
</tr>
<tr>
<td>v₃</td>
<td>(x₃, y₃, z₃)</td>
<td>e₄,₁</td>
</tr>
<tr>
<td>v₄</td>
<td>(x₄, y₄, z₄)</td>
<td>e₇,₁</td>
</tr>
<tr>
<td>v₅</td>
<td>(x₅, y₅, z₅)</td>
<td>e₅,₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>face</th>
<th>edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td>e₁,₁</td>
</tr>
<tr>
<td>f₂</td>
<td>e₃,₂</td>
</tr>
<tr>
<td>f₃</td>
<td>e₄,₂</td>
</tr>
</tbody>
</table>
3. DCEL

Example

<table>
<thead>
<tr>
<th>Half-edge</th>
<th>origin</th>
<th>twin</th>
<th>Incident Face</th>
<th>next</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_{3,1}</td>
<td>v_3</td>
<td>e_{3,2}</td>
<td>f_1</td>
<td>e_{1,1}</td>
<td>e_{2,1}</td>
</tr>
<tr>
<td>e_{3,2}</td>
<td>v_2</td>
<td>e_{3,1}</td>
<td>f_2</td>
<td>e_{4,1}</td>
<td>e_{5,1}</td>
</tr>
<tr>
<td>e_{4,1}</td>
<td>v_3</td>
<td>e_{4,2}</td>
<td>f_2</td>
<td>e_{5,1}</td>
<td>e_{3,2}</td>
</tr>
<tr>
<td>e_{4,2}</td>
<td>v_5</td>
<td>e_{4,1}</td>
<td>f_3</td>
<td>e_{6,1}</td>
<td>e_{7,1}</td>
</tr>
</tbody>
</table>
3. DCEL

- **Pros:**
  - All queries in $O(1)$ time
  - All operations are (usually) $O(1)$

- **Cons:**
  - Represents only manifold meshes
Geometry Data

- Vertex position
  - \((x, y, z, w)\)
  - in model space or screen space

- Vertex normal
  - \((n_x, n_y, n_z)\)

- Vertex color
  - \((r, g, b)\) or (diffuse, specular)

- Texture coordinates on vertex
  - \((u_1, v_1), (u_2, v_2), \ldots\)

- Skin weights
  - \((\text{bone}_1, w_1, \text{bone}_2, w_2, \ldots)\)

- Tangent and bi-normal
Topology Data

- Lines
  - Line segments
  - Polyline
    - Open / closed
- Indexed triangles
- Triangle strips/fans
- Surfaces
  - Non-Uniform Rational B-Spline (NURBS)
- Subdivision
Indexed Triangles

- **Geometric data**
  - **Vertex data**
  - $v_0, v_1, v_2, v_3, ...$
  - $(x, y, z, n_x, n_y, n_z, t_u, t_v)$
  - or $(x, y, z, c_r, c_g, c_b, t_u, t_v, ...)$

- **Topology**
  - Face $v_0 \ v_3 \ v_6 \ v_7$
    - right-hand rule for index
  - Edge table
Triangle Strips/Fans

Get great performance to use triangle strips/fans for rendering on current hardware.
Mesh in Unity

- Creating or modifying meshes from scripts
- For every vertex, there can be a normal, two texture coordinates, color and tangent.
- The triangle arrays are simply indices into the vertex arrays; three indices for each triangle.
- If your mesh has 10 vertices, you would also have 10-size arrays for normals and other attributes.
MeshFilter in Unity

- A class to access the Mesh of the mesh filter.
- Use this with a procedural mesh interface.
Mesh in Unity

- Building a mesh from scratch

```csharp
Vector3[] newVertices;
Vector2[] newUV;
int[] newTriangles;

void Start() {
    Mesh mesh = new Mesh();
    GetComponent<MeshFilter>().mesh = mesh;
    mesh.vertices = newVertices; // Should be assigned before triangle index
    mesh.uv = newUV;
    mesh.triangles = newTriangles;
}
```
# Mesh in Unity

## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colors</td>
<td>Vertex colors of the Mesh.</td>
</tr>
<tr>
<td>colors32</td>
<td>Vertex colors of the Mesh.</td>
</tr>
<tr>
<td>normals</td>
<td>The normals of the Mesh.</td>
</tr>
<tr>
<td>tangents</td>
<td>The tangents of the Mesh.</td>
</tr>
<tr>
<td>triangles</td>
<td>An array containing all triangles in the Mesh.</td>
</tr>
<tr>
<td>uv</td>
<td>The base texture coordinates of the Mesh.</td>
</tr>
<tr>
<td>uv2 ~ uv8</td>
<td>The second ~ eighth texture coordinate set of the mesh, if present.</td>
</tr>
<tr>
<td>vertexCount</td>
<td>Returns the number of vertices in the Mesh (Read Only).</td>
</tr>
<tr>
<td>vertices</td>
<td>Returns a copy of the vertex positions or assigns a new vertex positions array.</td>
</tr>
</tbody>
</table>
# Mesh in Unity

## Public Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clear</strong></td>
<td>Clears all vertex data and all triangle indices.</td>
</tr>
<tr>
<td><strong>CombineMeshes</strong></td>
<td>Combines several Meshes into this Mesh.</td>
</tr>
<tr>
<td><strong>GetColors</strong></td>
<td>Gets the vertex colors for this instance.</td>
</tr>
<tr>
<td><strong>SetColors</strong></td>
<td>Vertex colors of the Mesh.</td>
</tr>
<tr>
<td><strong>RecalculateNormals</strong></td>
<td>Recalculates the normals of the Mesh from the triangles and vertices.</td>
</tr>
<tr>
<td><strong>UploadMeshData</strong></td>
<td>Upload previously done Mesh modifications to the graphics API.</td>
</tr>
</tbody>
</table>
Surface Properties

- Material
- Textures
- Shaders
Materials

- **Material**
  - Ambient
    - Environment
    - Non-lighted area
  - Diffuse
    - Dynamic lighting
  - Emissive
    - Self-lighting
  - Specular with shininess
    - Hi-light
    - View-dependent
    - Not good for hardware rendering

- **Local illumination**
Textures

- Single texture
- Texture coordinate animation
- Texture animation
- Multiple textures
- Alphamap

Base color texture

Material or vertex colors

Lightmap
Shaders

- Programmable shading language
  - Vertex shader
  - Pixel shader

- Procedural way to implement some process of rendering
  - Transformation
  - Lighting
  - Texturing
  - BRDF
  - Rasterization
  - Pixel fill-in
  - Post-processing for rendering
Powered by Shaders

- Per-pixel lighting
- Motion blur
- Volume / Height fog
- Volume lines
- Depth of field
- Fur rendering
- Reflection / Refraction
- NPR
- Shadow
- Linear algebra operators
- Perlin noise
- Quaternion
- Sparse matrix solvers
- Skin bone deformation
- Normal map
- Displacement map
- Particle shader
- Procedural Morphing
- Water Simulation
Surface Properties in Unity

- Material
- Shaders
- Texture
Bounding Volumes

- Bounding sphere
- Bounding cylinder
- Axis-aligned bounding box (AABB)
- Oriented bounding box (OBB)
- Discrete oriented polytope (k-DOP)
Bounding Volume - Application

- Collision detection
- Visibility culling
- Hit test
- Steering behavior
  - in “Game AI”
Application Example – Bounding Sphere

Bounding sphere $B_1(c_1, r_1)$, $B_2(c_2, r_2)$

If the distance between two bounding spheres is larger than the sum of radius of the spheres, than these two objects have no chance to collide.

$D > \text{Sum}(r_1, r_2)$
Application Example - AABB

- Axis-aligned bounding box (AABB)
  - Simplified calculation using axis-alignment feature
  - But need run-timely to track the bounding box
Application Example - OBB

- Oriented bounding box (OBB)
  - Need intersection calculation using the transformed OBB geometric data
  - 3D containment test
  - Line intersection with plane

- For games, 😊
Colliders in Unity

- BoxCollider
- SphereCollider
- CapsuleCollider
- MeshCollider
- If the object with the Collider needs to be moved during gameplay then you should also attach a Rigidbody component to the object.
- The Rigidbody can be set to be kinematic if you don't want the object to have physical interaction with other objects.
Colliders in Unity
Colliders as Triggers in Unity

- Trigger events are only sent if one of the Colliders also has a RigidBody attached.
- Trigger events will be sent to disabled MonoBehaviours, to allow enabling Behaviours in response to collisions.
- Triggers are only supported on convex colliders.
Colliders in Unity

- **OnCollisionEnter**: OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.
- **OnCollisionExit**: OnCollisionExit is called when this collider/rigidbody has stopped touching another rigidbody/collider.
- **OnCollisionStay**: OnCollisionStay is called once per frame for every collider/rigidbody that is touching another rigidbody/collider.
- **OnTriggerEnter**: OnTriggerEnter is called when the Collider other enters the trigger.
- **OnTriggerExit**: OnTriggerExit is called when the Collider other has stopped touching the trigger.
- **OnTriggerStay**: OnTriggerStay is called almost all the frames for every Collider other that is touching the trigger. The function is on the physics timer so it won't necessarily run every frame.
- **OnCollisionEnter**: OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.
- **OnCollisionExit**: OnCollisionExit is called when this collider/rigidbody has stopped touching another rigidbody/collider.
Colliders in Unity

```csharp
void OnCollisionEnter(Collision collision) {
    // Show ContactPoint
    foreach (ContactPoint contact in collision.contacts) {
        Debug.DrawRay(contact.point, contact.normal, Color.white);
    }

    // Play a sound when a collision occurs
    if (collision.relativeVelocity.magnitude > 2)
        audioSource.Play();
}
```
Ray Casting

Center of projection

Window
Spatial Partitioning
Spatial Partitioning
Spatial Partitioning
Space Subdivision Approaches

- Uniform grid
- K-d tree
Space Subdivision Approaches

Quadtree (2D)  
Octree (3D)  
BSP tree
Uniform Grid
Uniform Grid

Preprocess scene
1. Find bounding box
Uniform Grid

Preprocess scene
1. Find bounding box
2. Determine grid resolution
Uniform Grid

Preprocess scene
1. Find bounding box
2. Determine grid resolution
3. Place object in cell if its bounding box overlaps the cell
Uniform Grid

Preprocess scene
1. Find bounding box
2. Determine grid resolution
3. Place object in cell if its bounding box overlaps the cell
4. Check that object overlaps cell (expensive!)
Uniform Grid Traversal

- Preprocess scene
- Traverse grid
- 3D line = 3D-DDA
From Uniform Grid to Quadtree
Quadtree (Octrees)

subdivide the space adaptively
Quadtree Data Structure

Quadrant Numbering

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Quadtree Data Structure

Quadrant Numbering

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing a tree structure with quadrant numbering.
Quadtree Data Structure

Quadrant Numbering

2 3
0 1

P

0 1 2 3

P P P P

P E F P

P P F F P

P P
Quadtree Data Structure

Quadrant Numbering

0 1
2 3

Diagram of a quadtree with quadrant numbering.

78
From Quadtree to Octree
Leaf nodes correspond to unique regions in space
K-d Tree

Leaf nodes correspond to unique regions in space
K-d Tree

Leaf nodes correspond to unique regions in space
K-d Tree
K-d Tree
K-d Tree
K-d Tree
Leaf nodes correspond to unique regions in space
Leaf nodes correspond to unique regions in space
Binary Space-Partitioning Trees
Binary Space-Partitioning Trees
Splitting triangles
BSP Tree
BSP Tree
BSP Tree

point
BSP Tree Traversal
BSP Tree Traversal
Level-of-Details

- Discrete LOD
  - Switch multiple resolution models run-timely

- Continuous LOD
  - Use progressive mesh to dynamically reduce the rendered polygons

- View-dependent LOD
  - Basically for terrain
Level of Detail: The Basic Idea

- One solution:
  - Simplify the polygonal geometry of small or distant objects
  - Known as *Level of Detail* or *LOD*
    - a.k.a. polygonal simplification, geometric simplification, mesh reduction, multiresolution modeling, ...
Level of Detail: Traditional Approach

- Create *levels of detail (LODs)* of objects:

  69,451 polys  2,502 polys  251 polys  76 polys
Level of Detail: Traditional Approach

- Distant objects use coarser LODs:
Traditional Approach: Discrete Level of Detail

- Traditional LOD in a nutshell:
  - Create LODs for each object separately in a preprocess
  - At run-time, pick each object’s LOD according to the object’s distance (or similar criterion)

- Since LODs are created offline at fixed resolutions, this can be referred as Discrete LOD
Discrete LOD: Advantages

- Simplest programming model; decouples simplification and rendering
  - LOD creation need not address real-time rendering constraints
  - Run-time rendering need only pick LODs
Discrete LOD:
Advantages

- Fits modern graphics hardware well
  - Easy to compile each LOD into triangle strips, display lists, vertex arrays, …
  - These render much faster than unorganized polygons on today’s hardware (3-5 x)
Discrete LOD: Disadvantages

- So why use anything but discrete LOD?
- Answer: sometimes discrete LOD not suited for *drastic simplification*
- Some problem cases:
  - Terrain flyovers
  - Volumetric isosurfaces
  - Super-detailed range scans
  - Massive CAD models
Continuous Level of Detail

- A departure from the traditional static approach:
  - Discrete LOD: create individual LODs in a preprocess
  - Continuous LOD: create data structure from which a desired level of detail can be extracted at run time.
Continuous LOD: Advantages

- Better granularity $\rightarrow$ better fidelity
  - LOD is specified exactly, not chosen from a few pre-created options
  - Thus objects use no more polygons than necessary, which frees up polygons for other objects
  - Net result: better resource utilization, leading to better overall fidelity/polygon
Continuous LOD: Advantages

- Better granularity $\rightarrow$ smoother transitions
  - Switching between traditional LODs can introduce visual “popping” effect
  - Continuous LOD can adjust detail gradually and incrementally, reducing visual pops
- Can even *geomorph* the fine-grained simplification operations over several frames to eliminate pops [Hoppe 96, 98]
Continuous LOD: Advantages

- Supports progressive transmission
  - Progressive Meshes [Hoppe 97]
  - Progressive Forest Split Compression [Taubin 98]

- Leads to view-dependent LOD
  - Use current view parameters to select best representation for *the current view*
  - Single objects may thus span several levels of detail
Methodology

- Sequence of local operations
  - Involve near neighbors - only small patch affected in each operation
  - Each operation introduces error
  - Find and apply operation which introduces the least error
Simplification Operations

- Decimation
  - Vertex removal
    - $v \leftarrow v-1$
    - $f \leftarrow f-2$
  - Remaining vertices - subset of original vertex set
Simplification Operations

- Decimation
  - Edge collapse
    - $v \leftarrow v-1$
    - $f \leftarrow f-2$
  - Triangle collapse
    - $v \leftarrow v-2$
    - $f \leftarrow f-4$

- Vertices may move
Simplification Error Metrics

- Measures
  - Distance to plane
  - Curvature

- Usually approximated
  - Average plane
  - Discrete curvature

\[ \sum \frac{\alpha}{2\pi} \]
The Basic Algorithm

- Repeat
  - Select the element with minimal error
  - Perform simplification operation
    - (remove/contract)
  - Update error
    - (local/global)

- Until mesh size / quality is achieved
Progressive Meshes

- Render a model in different Level-of-Detail at run time
- User-controlledly or automatically change the percentage of rendered vertices
- Use collapse map to control the simplification process

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Collapse map

<table>
<thead>
<tr>
<th>Vertex list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Triangle list</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Map of collapse process:
LOD in Unity

- Mesh setting
  - Setting each level of mesh renderer
LOD in Unity

- Mesh setting
  - Setting each level of mesh renderer
LOD in Unity

- Mesh setting
  - Setting each level of mesh renderer
LOD in Unity

- Terrain setting