Level of Details – A First Introduction

Richard (Hao) Zhang
CMPT 464/764: Geometric Modeling in Computer Graphics
Lecture 6
Level of details (LOD)

Level of details or LODs provide a way to manage scene complexity.

**Goal:** Render as few triangles — reduce triangle throughput — as necessary based on quality demand and/or available resources.

GPU changes this somewhat …
What is LOD?

- Scene is described at **multiple Levels of Detail**
- Such a model description can be
  - **Pre-computed** — discrete LOD, or
  - **Obtained at run time** — continuous LOD, e.g., for a virtual walk
- a. k. a. polygonal/geometry simplification, mesh reduction, mesh decimation, multi-resolution modeling, etc.
- LODs are typically chosen based on
  - Application’s demand for visual quality — **fidelity-based**, and/or
  - Available hardware/computational resources — **budget-based**

Meshlab demo
An optimization problem at heart

Fundamental questions:

- Given a particular quality criteria, how to fulfill it with the least number of triangles?
- Given a particular triangle count, e.g., one that is allowed by the current bandwidth restrictions, how to maximize the fidelity of your geometric representations?
- In most case, the global optimization problem is intractable
- Approximation scheme, greedy algorithms, lazy computation schemes, etc., are necessary
Some key questions

- How to measure approximation quality or **simplification error**?
- How to get **smooth transition** between LODs – no “popping”?
- **Speed, speed and more speed**: efficient algorithms to achieve interactivity to serve computer games, virtual walkthrough, etc.
- **Utilization of GPU** (modern computer graphics hardware) — our thinking changes a little bit …

Now a standard tool for the graphics practitioner, e.g., in heavy use in computer and video games, simulation applications
LOD vs. subdivision

- Subdivision generates LOD models through refinement.
- But stationary subdivision schemes does not increase the information content of the model.

Loop  Butterfly  Camull-Clark  Doo-Sabin
To model specific geometry using subdivision, need to add **extra shape-specific details** after each step.

Meshes obtained have **subdivision connectivity**.
For meshes with **arbitrary connectivity**, LODs are obtained through **incremental mesh decimation**.

Mesh primitives – vertices, edges, and/or faces – are removed one or many at a time:

- What and how to remove?
- How to modify mesh after removal?
- Coarse version should have good approximation quality – measured by some simplification **error metric**
Idea of LOD is everywhere

- Use of crude, often hierarchical, approximations
  - Subdivision surface modeling
  - Quantization – one way for data compression
  - Mesh simplification and refinement
  - Mipmaps in texture mapping … GeoMipMaps in terrain rendering
  - Image pyramids – convolutional neural networks (CNNs)
  - Quadtree, octree hierarchical bounding volumes for fast view frustum culling, back face culling, silhouette extraction, etc.
  - Collision detection using convex hulls, etc.
We shall focus on …

**LODs on geometry of meshes** (a little bit on **topology LOD**)

69,451 polys  
2,502 polys  
251 polys  
76 polys

Courtesy Stanford 3D Scanning Repository
Why LOD?

- **Wealth of applications**: interactive rendering of large-scale geometric datasets is of great practical importance
  - Scientific and medical visualization
  - Architectural and industrial CAD
  - Training (military and others, e.g., flight simulation)
  - Entertainment, esp. **computer games**
  - Art historical study
  - Etc.
Why LOD?

- **High complexity:**
  - With the advent and advances in 3D scanning and acquisition technology, complex polygon models are commonplace
  - Many such models are too complex to render at interactive rates
- **Even worse:**
  - Incredibly, model complexities seem to grow faster than hardware capacities — *a reason to do research 😊*
- **Never-ending demand for realism**
  - Advanced lighting, texturing, etc., to share limited computational resources — so we need to “squeeze out the triangles” …
Big models

- Submarine torpedo room: 700,000 polygons

Courtesy General Dynamics, Electric Boat Div.
Big models

- Coal-fired power plant: 13 million polygons
Big models

Plant ecosystem simulation: 16.7 million polygons

Deussen et al: Realistic Modeling of Plant Ecosystems
Big models

- Double eagle container ship: 82 million polygons

Courtesy Newport News Shipbuilding
Big models

- The Digital Michelangelo Project
  - David: 56,230,343 polygons
  - St. Matthew: 372,422,615 polygons

Courtesy Digital Michelangelo Project
Big models

- Pietà project of IBM Research
  - Scan of Michelangelo’s Florentine Pietà for art historical study
  - Took 90 hours over 14 days
  - 800 separate scans
  - Raw data file size: 3 GB

[Bernardini et al. 01]
Traditional approach to LOD

- Small or distant objects use coarser LODs
- LOD for graphics originally proposed in 1976 by Clark – whose system setup still in use in today’s systems due to its simplicity
Framework 1: discrete LOD

- Create and store **discrete LODs for each object** separately in **offline preprocessing** – LOD does not exist for each polygon count

- At run-time, pick each object’s LOD according to its size, distance to viewer, or other criterion

- One advantage: simplification and rendering/processing decoupled
  - LOD creation needs not address real-time rendering constraints
  - Run-time rendering needs only pick LODs – efficient
  - They can be **separately optimized**
More advantages

- Offline LOD construction can focus on approximation quality — take as long as it would take to optimize

- Better utilization of GPU

  - Can compile these static LOD models into long triangle strips or other more optimized rendering sequences to increase cache performance

  - These can be rendered much faster than unorganized polygons on today’s hardware (3-5 x or more)
Triangle strips

- A triangle strip gives a compact way of representing a set of triangles.
- For $n$ triangles in a strip, instead of passing through and transform $3n$ vertices, only need $n+2$ vertices.
- In a sequence, e.g., $v_1, v_2, v_3, v_4, \ldots$, first three vertices form the first triangle; each subsequent vertex forms a new triangle with its preceding two vertices.
- Many algorithms exist to “stripify” a triangle mesh into long triangle strips.
Optimized rendering sequences

- Key is to maximize **cache hits**
  - Hierarchical memory structure (cache – very fast access time but small)
  - Current GPU typically has a vertex cache that holds 20+ vertices but it is variable depending on how much info is needed per vertex

- Highly desirable for a rendered triangle to **already have one or more vertices in the cache** – cache hits; otherwise cache misses

- Measurement of performance:
  \[
  \text{ACMR} = \frac{\text{total # cache misses}}{\text{total # triangles}}
  \]

- Maximum possible ACMR: 3; minimum possible ACMR: about 0.5
More on ACMR and face sequences

- ACMR depends on **cache size** and **cache replacement policy**
  - Random
  - FIFO: first in first out – most frequently optimized
  - LFU: least frequent used (out)
  - LRU: least recently used – better performance but hard to optimize for
- Also desirable to have a **cache-oblivious** approach
  - Good cache performance regardless of the cache parameters
  - Intuitive idea: traverse ALL faces around a vertex over short span in the sequence covered by cache — sequence with **good locality** property
  - Adjacent faces on mesh should be close in the sequence
E.g., fractal-like space-filling curves

Hilbert curves: can be generalized to arbitrary triangle meshes
Disadvantages of discrete LOD

- **Not enough flexibility or granularity**
  - Not enough levels of details
  - The most appropriate polygon counts may fall in-between two LODs
- Not suitable for very complex scenes, as in
  - Terrain flyovers, volumetric iso-surfaces,
  - Super-detailed range scans, massive CAD models, etc.
- **Switching between two levels is not continuous**
E.g., A really large model

Courtesy IBM and ACOG
Framework 2: continuous LOD

… or dynamic LOD or progressive LOD:

- Data structure allows desired LOD to be extracted at run time

- Advantages:
  - **Better granularity** – appropriate polygon count determined by fidelity requirement or computational budget continuously over time
  - **Better use of resources** – get only what is needed … no overshooting or undershooting
  - Support **progressive transmission** and display
More pros and … cons

- Better granularity leads to **smoother transitions**
  - Switching between discrete LODs introduces visual “popping”
  - Continuous LODs adjust detail gradually and incrementally
  - Can even **geomorph** the fine-grained simplification operations over several frames to completely eliminate popping further

- Support for **view-dependent LODs**

- Main disadvantage:
  - **Complex algorithms** and **computationally demanding**
Framework 3: view-dependent LOD

- Basically divides large object into different LOD regions
- **Anisotropic:** show nearby regions to the viewer of objects/scenes at higher resolution than distant portions

View from eye point  
Birds-eye view
View-dependent LOD: Silhouette

- Typically show **silhouette region** of an object at higher resolution than interior region – human eyes are sensitive to edges
View-dependent LOD: Eccentricity

- Show more detail where user is looking than in their peripheral vision

34,321 triangles
View-dependent LOD: Eccentricity

- Show more detail where user is looking than in their peripheral vision

11,726 triangles
View-dependent LOD: pros and cons

- **Pro**: even better granularity
  - Allocates polygons where they matter to viewing
  - Enables even better overall fidelity and utilization of resources
- **Pro**: allows drastic simplification of very large objects
- **Main disadvantages**: 
  - Even more computationally demanding, e.g., need to keep track of continuous view changes, e.g., eye tracker
  - Even more complex algorithms
Hierarchical LOD

- View-dependent LOD suitable with single large object
- Hierarchical LOD suitable with many small objects in the scene
  - Use a hierarchical data structure, e.g., an octree
  - Merge nearby objects into assemblies
  - At a distance, simplify assemblies, not individual objects
- Hierarchical LOD implies a topology-modifying algorithm
LOD in practice

- Discrete LOD still quite common, especially in computer games
  - Simple programming
  - Better utilization of GPU – very important nowadays

- Continuous and view-dependent LOD – higher cost
  - More complex data structures
  - Cost to evaluate, simplify, and refine at run time
  - Less GPU-friendly: there is a recent shift of goals from reducing triangle throughput to producing triangles best suitable for GPU processing
  - Some continuous LOD algorithms aim to provide good trade-off
A special case: terrain LOD

- Wide-ranging applications
  - Flight simulation
  - Computer games
  - Geographic information systems (GIS) – emergence of LiDAR capture
  - Mission planning in the military training, etc.

- A simpler and special case of generic LOD
  - More constrained geometric domain
  - Normally a height field over a regular grid, like an image – a. k. a. DEM (digital elevation models)
  - More specialized and often simpler algorithms
Characteristics

- Conceptually simpler but can be **HUGE**, e.g., 1.8 billions triangles in a US Geological Survey dataset
- May have a **large amount of terrain visible** at many views that recedes far into the distance (compare to a bunny, for example)
- **View-dependent LOD** is critical for real-time systems
- Represents the earliest work on LOD (late 70’s)
An example


Courtesy of Alex Pfaffe

RTIN grid overlaid on terrain surface

Performance measures

LOD depiction
Notable issues

- **Top-down vs. bottom-up** LOD construction
  - In other words, refinement/subdivision vs. decimation

- Grid structure adopted in representation: regular grids vs. **TINs** (Triangulated irregular networks) vs. hybrid

- **Adaptive** multiresolution structure: Quadtree vs. bintree

- Artifacts to avoid: **cracks and T-junctions**
Regular grids

- Uniform array of height (z) values — less memory
- Easy to encode in raster formats, e.g., DEM (digital elevation models), GeoTIFF, etc. — easy compression
- Natural hierarchical representation (e.g., using mipmaps)
- But have obvious disadvantages
  - **Sub-optimal in triangle count** (not adaptive)
  - Generally ineffective in modeling sharp, e.g., ridge, features
**TINs**

- Triangulated Irregular Networks
- Fewer triangles needed to attain required accuracy, but need to \((x, y, z)\) per vertex
- Higher sampling around bumpy regions and less samples around flat regions
- More flexibility in modeling maxima, ridge lines, etc.
- But more complex algorithms, e.g., when constructing hierarchies
- **Hybrid approaches** exist, e.g., RTINs or bintrees, [Evans 97]
Multiresolution structures

Quad-trees

Bin-trees

Require \textit{refinability}
T-junctions and cracks

- Cracks occur when different LODs join

Cracks – holes in rendered terrain

T-junctions – **bleeding tears** due to finite floating-point (object) precision

May not be able to lie on edge
Avoiding T-junctions via subdivision
Avoiding T-junctions via subdivision
Avoiding T-juctions via subdivision
Avoiding T-junctions via subdivision
What are next?

- **Simplification error metrics**
  - Object-space geometric errors
  - Image-space errors
  - A little bit perceptual considerations

- **Mesh simplification**: algorithms and strategies