Lazy Incremental Computation for Efficient Scene Graph Rendering

Michael Wörister, Harald Steinlechner, Stefan Maierhofer, and Robert F. Tobler

VRVis Research Center
Vienna, Austria
(Hierarchical) Scenes modelled Scene Graphs

Regular scene graph rendering algorithm can become inefficient

- for large number of nodes / paths

Profiling shows: much time spent with traversal overhead

- matrix multiplications
- virtual function calls
- ...

→ scene graph traversal becomes performance bottleneck
Scene graph Optimizations

Sorting
Packing
Batching
Instancing

Harald Steinlechner
Optimizations affect Model/Design

Optimizations leaks into application
- Mutation of modeling datastructure ruins clean semantic view of the scene

Interaction of optimization with dynamic scenes
- Some optimizations not valid in general (blending)
- Expensive geometry/texture packing

How to combine with dynamic scenes? CONFLICT
Separate Model/Optimization

- Retain original datastructure
- Additional optimization datastructure

Keep optimization datastructure in sync!

Instruction-Array

GPU-Resources
Efficient Synchronization

- Changes in Scene graph
- Modification to the tree or attributes
- Change propagation in: $O(|\text{AFFECTED}|)$
- In-place updates / Structural updates
- In this work: fast In-place updates
Incremental Computation

- Given input $x$ and $f(x)$, find changes of $f(x)$ given changes in $x$
- Originally used for Attribute Evaluation for Attribute Grammars
- Builds on static dependency Graph
Dependency Graphs

Dependency Graphs used in

- Build systems (like *make*)
- Compilers (Data/Flow dependencies)
- Visual programming (like Hypergraph, Hypershade)
The Implied Dependency Graph

- Geometry node → Leaf node
- Dependency in Sg → Dependency Node
- Computation → Dependency Node
Towards lazy attribute evaluation

- *Standard Optimal Algorithm* [Reps et al. 1983] not suitable
- Scene graphs are DAGs, parts may be culled
- Demand driven approach by [Hudson 1991]
Towards lazy attribute evaluation

- Step 1: Dependency triggers, perform out of date marking
- Step 2: Update required values (recompute nodes which are out of date)
- Step 3: **Render**
Not yet there: marking is eager

Large parts not visible
- Marking not necessary/feasable
- Replace eager marking with **lazy polling**

Keep list of transitive reachable Dependencies
- Check for all predicates directly at cache entry

\[ r_1 \rightarrow \{ d_1, d_2, \ldots, d_n \} \]
Building an incremental Render Cache

Create Dependency Index data structure for cache entries affected by change

Based on type, this entry knows how to update the cache entry using remembered scenegraph nodes
Dependency Index

Inverted index data structure

- Maps dependency objects to list of their dependent resources
- Untouched dependencies checked once per cache
- Changed resources touched once, updates version number
Solid Foundation for Optimizations

For static scenes
- State Sorting
- Removal of redundant instructions
- „Super Instructions“
- Generalized Draw Sorting

For dynamic scenes
- Parallel Cache Update
- Memoized Transformation Matrices
Evaluation: Worst case

Simulate worst case

- Distinct buffers, distinct draw calls
- Different shaders, materials etc.

Many, many draw calls

- Draw call reduction not sufficient (culling)
- Everything is visible
- Huge dependency graph

GPU load static (poly counts)
Static Scenes

![Graph showing frame time vs. draw calls for static scenes with uncached and cached renderings. The graph has a linear relationship with a significant time penalty for increasing draw calls.](image)

- **Label**: Frame Time (ms)
- **X-axis**: Draw Calls (thousands)
- **Y-axis**: Frame Time (ms)

Legend:
- **Uncached**
- **Cached**

The graph illustrates the performance difference between rendering static scenes uncached and cached, highlighting the efficiency gains of caching for high draw call scenarios.
Optimizations: Factor 2.5

Frame Time (ms)

- No Caching
- Caching
- Redundancy Removal
- State Sorting
- Super-Instructions
- All
CPU Optimizations: Huge improvement

Frame Time (ms)

- No Caching
- Caching
- Redundancy Removal
- State Sorting
- Super-Instructions
- All

>168
What are the costs?

- Test scene with 22k objects, 224MB memory, 669MB graphics memory
- Additional 3MB main memory (dependencies) + additional 3MB graphics memory for caching (buffers)
OpenSceneGraph Comparison

![Graph showing comparison between cached-optimized and OpenSceneGraph in terms of frame time and draw calls]

- **Cached-optimized**
- **OpenSceneGraph**
Dynamic Scene Setup

Octree structure
- 2 trafo nodes each level
- depth: 5, some leafs empty

Percentage of dynamic objects
- randomized, some trafos dynamic
- varying from 0 – 100 percent changing trafo nodes
Dynamic Scenes

![Graph showing frame time vs. percentage of dynamic geometries for different scenarios: cached-optimized, cached-optimized-parallel, OpenSceneGraph, and OpenSceneGraph, multithreaded. The graph illustrates performance improvements with increasing percentage of dynamic geometries.]
Future Work

Achieved: efficient rendering of high level scene graph

Structural scene graph changes
- add/remove/change arbitrary nodes
- caches need to built from scratch for this type
- improve/generalize incremental model

Improved runtime system
- optimization at runtime / on demand
- automatic placement of render caches
Thank you for your attention!

Please visit us at

http://www.VRVis.at/
Static Scenes (2)

![Graph showing frame time (ms) vs draw calls for uncached and cached cases. The graph demonstrates a linear increase in frame time with increasing draw calls for the uncached case, while the cached case remains relatively flat.]
Semantic Scenegraph

- Each Node is expanded into the rendering scene graph
- One additional layer of indirection.

[Semantically structured data visualization example with a semantic scene graph and a rendering scene graph, illustrating the process of translation between the two.]

[Tobler, 2011]
Incremental Computation in Computer Graphics (1)

Scene graph systems

- Open Inventor [Wernecke 1993], OpenSceneGraph [Burns and Osfield 2004], Java3D, NVIDIA Scenix
- Display list as caching

Hierarchical Caching

- [Durbin et al. 1995] render calls in main memory
- Redundancy Removal each frame
- Statistical Data – will cache amortize over time?
Incremental Computation in Computer Graphics (2)

Windows Presentation Foundation
- Dependency Graph for update propagation
- Not incremental

Maya Dependency Graphs
- Eager marking, Lazy Evaluation (similar to [Hudson 1991])
- No indication for caching (Maya internals)

Lighting/Shading
- Constant caching in Shading (for example Pixar Irma)
- Recently Lightspeed system [Ragan-Kelley et al. 2007]
Dependencies

Formalized Model with three primitives

- Value Sources (inputs from scene graph)
- Dependent Resources (cache entries derived from scene graph)
- Dependencies (predicates that describe change)

\[ d_1 = \text{user input} \]
Incremental Caching

Cached Rendering Algorithm

(1) Check for changes in scene graph
   - using the DependencyIndex data structure
   - yields a list of inconsistent cache entries

(2) Update cache where necessary
   - by triggering updates in affected cache entries
   - the meta data entry knows how to perform update

(3) Execute instructions
Detailed Test Setup

- Intel Core i7-3770 @ 3.4 GHZ
- NVIDIA GeForce GTX680, 2GB
- 32GB RAM

- screen resolution: 1920 x 1200
- no multisampling
Title of First Slide

A subheading [no bullet: in order to indent to introduce a bullet, click on the Paragraph indent button in the tool bar. As soon as you have a bullet you can use (shift-)Tab to (de-)indent]

- replace the presenter name in the master slide with your name
- and replace short title with your title (also on the master slide)

Another heading (no bullet: start with capital letter)

- start text after bullets with lower case
- use catch phrases
- do not use sentences ending with a full stop
Many Hierarchy Levels

Level 1
- level 2
  - try to avoid going to this level
    - this is even worse
      - and this is positively ridiculous
This Slide is an Example For a long Two-Line Title

- a list of bullets (just use indent)
- a list of bullets (just use indent)
- a list of bullets (just use indent)
- a list of bullets (just use indent)
- a list of bullets (just use indent)
- a list of bullets (just use indent)