Hardware-Compatible Vertex Compression Using Quantization and Simplification

Budirijanto Purnomo, Jonathan Bilodeau,
Jonathan D. Cohen and Subodh Kumar

Johns Hopkins University
Department of Computer Science

Motivation

- Big models (huge geometry, large texture data, complex shaders)
- Fast rendering using video memory
  - 4x faster than main memory
- Improving bottlenecks
  - Bus bandwidth
  - Restricted video memory size
- Little support for vertex data compression

Problem Statement

Vertex compression technique suitable for efficient decompression on graphics hardware.

3D model representation:

- Vertex data (and attributes)
- Triangle index list (connectivity information)

New Tools

We provide solutions that maximize rendering quality for the following problems:

- Given target bits per vertex (BPV), compute bit allocations to vertex attributes.
- Given target total storage, compute bit allocations and the simplified mesh.

Overview

Current graphics hardware constraints:

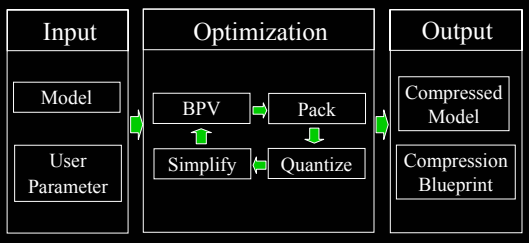
- Each vertex processed independently without side effects.
- Data channels limited to those exposed by current APIs.
- Vertex data should be multiple of 32 bits.
- No supports for native integer operations on vertex program.

Previous Work

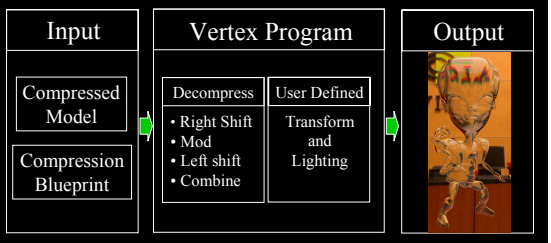
Vertex data compression

- Prediction: Deering 1995, Taubin 1995
- Spectral technique: Karni and Gotsman 2000
- Basic principles for dealing with quantization in vertex shader: Calver 2002 and 2004

System Overview - Preprocessing



System Overview - Runtime



Bit Allocation

Goal:

"Optimize allocation of bits to each vertex attribute component (*x*, *y*, *z*, *nx*, *ny*, *nz*, *r*, *g*, *b*, …) subject to the target bits per vertex and the error metric."

Image-Space Error Metric

Based on Lindstrom and Turk 2000's work on simplifying 3D meshes.

- Quantify error of allocating bits to multiple attributes (position vs normals vs colors) automatically.
- Account for any desired rendering algorithm and shader.

Image-Space Error Metric (cont.)

Algorithm (*n* = 20):

- Render *n* images from *n* viewpoints.
- Compare *n* images of altered models to renderings of original model.
- RMS of difference images.
- Do not consider background pixels.

Quantization

Vertex attributes:

- Position
- Normals
- Texture coord.

RocketCar model courtesy of NVIDIA Corporation

Quantization (cont.)

Original (896 KB)
(xyz, normal, texcoord) =
(32 32 32 32 32 32 32 32)
Error = 0.0

Our result at 64 bpv (224 KB)
(xyz, normal, texcoord) =
(11 11 12 5 5 5 7 8)
Error = 0.018

Quantization (cont.)

Our result at 64 bpv (224 KB)
(xyz, normal, texcoord) = 11 11 12 5 5 5 7 8
Error = 0.018

DirectX at 96 bpv (336 KB)
(xyz, normal, texcoord) = 10 10 10 10 10 10 16 16
Error = 0.021

Quantization (cont.)

Our result at 64 bpv (224 KB)
(xyz, normal, texcoord) = 11 11 12 5 5 5 7 8
Error = 0.018

OpenGL at 96 bpv (336 KB)
(xyz, normal, texcoord) = 8 8 8 8 8 8 16 16
Error = 0.040

Bit Allocation Algorithm

Input: BPV

- Initialize bit allocation
- Iterate until error no longer decreases
  - Increase
    - For each attribute component, increase one bit and measure current error
    - Pick the one with the lowest error.
  - Decrease
    - For each attribute component, decrease one bit and measure current error
    - Pick the one with the lowest error.

Packing

RocketCar 64 bpv

- (xyz, normals, texcoord) = 11 11 12 5 5 5 7 8
- Stored into a vectorized data element of 4 unsigned shorts.

[0]			[1]			[2]		[3]	
x	y	z	nx	ny	nz	u	v		

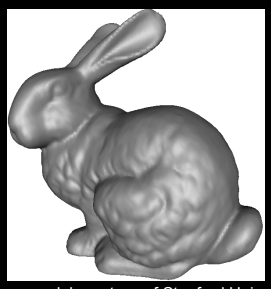
Combining Quantization and Simplification

Input: Target model size

Goal:

"Compute quantization and simplification that maximize the rendering quality for the specified size"

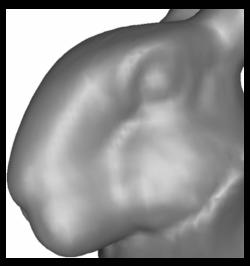
Combining Quantization and Simplification (cont.)



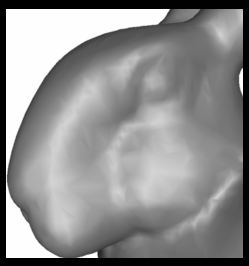
Vertex attributes:
- Position
- Normals

Bunny model courtesy of Stanford University

Combining Quantization and Simplification (cont.)

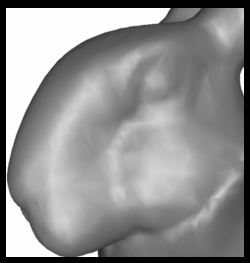


Original
(816 KB)
Error = 0.0

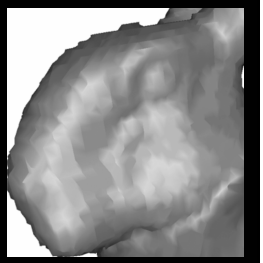


Quantized and Simplified
(136 KB)
Error = 0.016

Combining Quantization and Simplification (cont.)

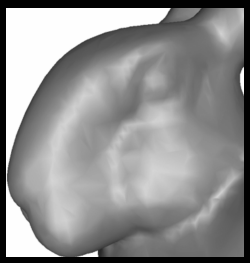


Quantized and Simplified
(136 KB)
Error = 0.016

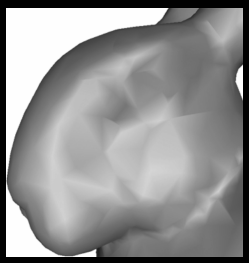


Quantized only
(136 KB)
Error = 0.056

Combining Quantization and Simplification (cont.)

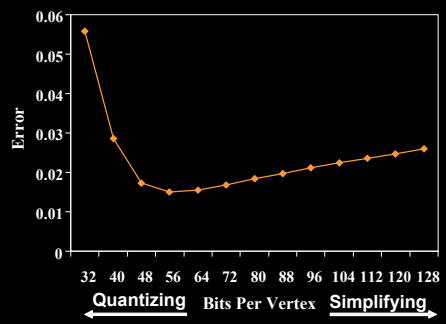


Quantized and Simplified
(136 KB)
Error = 0.016



Simplified only
(136 KB)
Error = 0.033

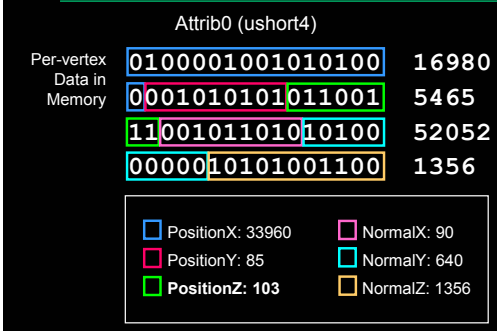
Combining Quantization and Simplification (cont.)



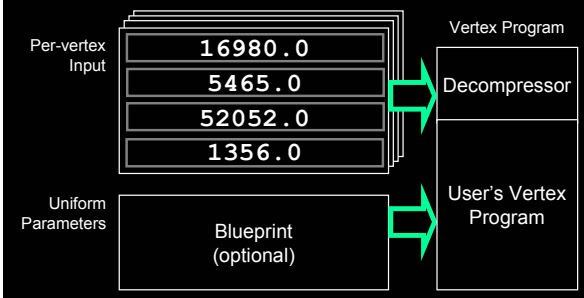
Combining Quantization and Simplification (cont.)

Algorithm:
- Compute current BPV from # verts.
- Compute bit allocation and error.
- Iterate:
  - Increase BPV (hence decrease # verts)
  - Simplify model based on new # verts
  - Compute bit allocation and error
  - Record best bit allocation and model so far
- Return best bit allocation and simplified model.

Storing Packed Data



Storing Packed Data

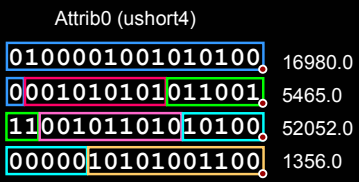


Decompression

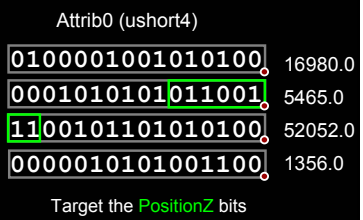
In Decompressor

- Receive packed, per-vertex data
- Extract bits for each vertex attribute component
- Assign to variables defined by existing vertex program

Extracting Bits

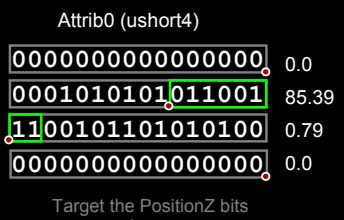


Extracting Bits



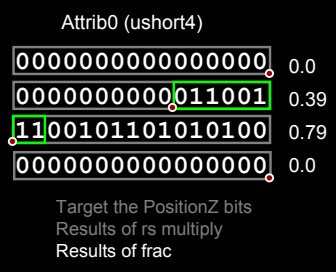
Extracting Bits

1. Right Shift
   - Multiply by *rs* component of the blueprint.
   - Moves the decimal point to the left of target bits.



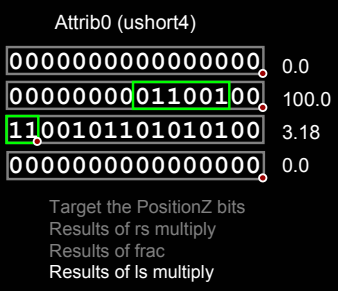
Extracting Bits

2. Frac
- Frac function returns the fractional part of a float.
- Zeros the bits to left of the target bits.



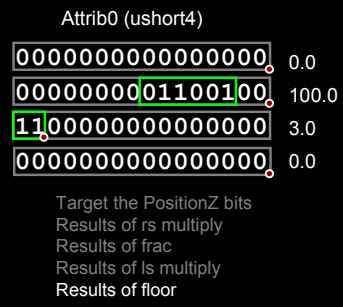
Extracting Bits

3. Left Shift
- Multiply by *ls* component of the blueprint.
- Shifts the bits to their original significance.



Extracting Bits

4. Floor
- Zeros the bits to the right of the target bits.



Decompression

Two Approaches
- General vertex program
  - Single program compatible with all bit layouts
  - Blueprint defined by uniform parameters
- Custom vertex program
  - New program generated for particular bit layout
  - Blueprint embedded in program logic
  - Potentially much smaller program length

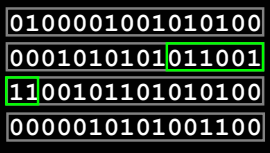
General Program

Extraction (For each attribute):
- Perform extraction process for each attribute component
- Sum results
  - effectively a logical OR

General Program

Sample Cg Code

```
results = floor(lsz*frac(rsz*bin));
results.xy = results.xy+results.zw;
pos.z = results.x+results.y;
```



Custom Program

Program written for a specific layout

Optimized general program

- Simultaneous extraction of multiple attribute components
- `frac` and `floor` can be optimized
- Blueprint is hard coded

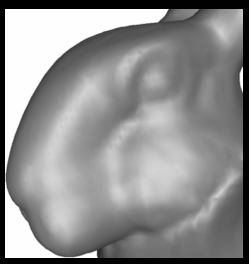
Custom Program

Sample Cg Code

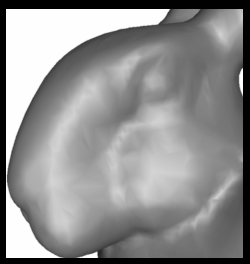
```
results = ls1*frac(rs1*bin1);
pos.x  += results.x;
pos.z  += results.y;
norm.y += results.z;
norm.z += results.w;
```

```
0100001001010100
0001010101011001
1100101101010100
0000010101001100
```

Timing Comparisons

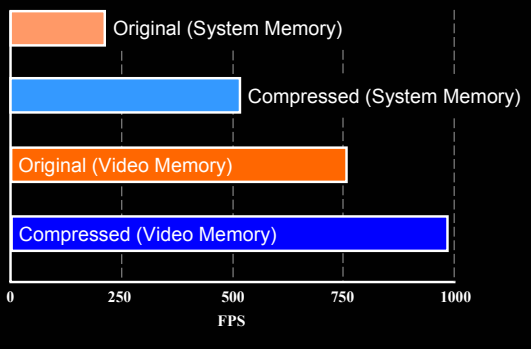


Original
(816 KB)

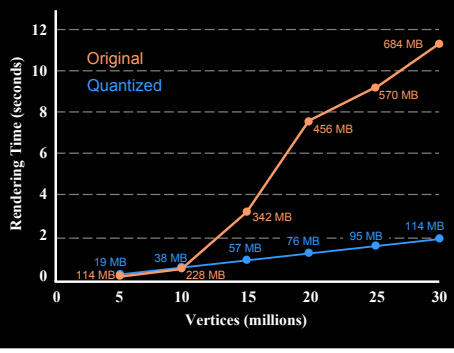


Quantized and Simplified
(136 KB)

Timing Comparisons



Quantization Only Timing



Possible Hardware Support

- Add decompression unit prior to vertex unit
  - Integer registers
- Pipelined with vertex unit
  - Minimal performance cost

Future Work

Exploiting graphics hardware for quantization-level computations

Extending the error metric

- Sampling across user parameters
  - Lighting conditions
  - Articulated models
  - Multiple viewing distances
    - Multiple levels of quantization

Conclusion

What we have presented

- A vertex compression scheme for graphics hardware
  - Optimizes visual quality for target model/vertex size
  - Allows decompression in a vertex program
- Two decompression options
  - A general procedure that can map well to hardware.
  - A custom scheme that is optimized for a bit layout and works well in software