Software Renderer Accelerated by CUDA Technology

Wojciech Sterna

25th of January, 2011

Fundamental Concepts

Fundamental Concepts

- rendering — process of generating a 2D image from a 3D data set

Fundamental Concepts

- rendering — process of generating a 2D image from a 3D data set
- the graphics pipeline — set of stages the input 3D data set is passed through to generate the 2D output image

Fundamental Concepts

- rendering — process of generating a 2D image from a 3D data set
- the graphics pipeline — set of stages the input 3D data set is passed through to generate the 2D output image
- CUDA — technology from NVIDIA that allows for parallel general-purpose code execution using graphics cards

Goals of the Work

Goals of the Work

- implement in software a selected subset of OpenGL — Vainmoinen

Goals of the Work

- implement in software a selected subset of OpenGL — Vainmoinen
- speed the implementation with CUDA

Goals of the Work

- implement in software a selected subset of OpenGL — Vainmoinen
- speed the implementation with CUDA
- compare the performance of a reference application using:
  - Vainmoinen without CUDA
  - Vainmoinen with CUDA
  - OpenGL

World Definition

World Definition

- the world is built with triangles only
- every triangle consists of 3 vertices
- every vertex has a position, color and texture coordinate

Vertex Processing

Vertex Processing

- input vertex $P = (x, y, z)$ goes through a series of matrix transformations resulting in vertex $P' = (x', y', z')$, where:
  - (x', y') — position of the vertex on the screen in window coordinates
  - z' — distance of the vertex to the camera in normalized $[-1, 1]$ range

Vertex Processing

- input vertex $P = (x, y, z)$ goes through a series of matrix transformations resulting in vertex $P' = (x', y', z')$, where:
  - (x', y') — position of the vertex on the screen in window coordinates
  - z' — distance of the vertex to the camera in normalized $[-1, 1]$ range

Other operations that take place during vertex processing:

- clipping to the near plane
- view frustum culling
- backface culling

Pixel Processing

Pixel Processing

- loop through the pixels of every triangle
- use barycentric coordinates to get the interpolated values (color and texture coordinate) at every pixel, given the values at the vertices of the triangle being processed

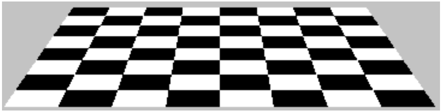
Pixel Processing

Naive interpolation leads to incorrectly rendered image:

Pixel Processing

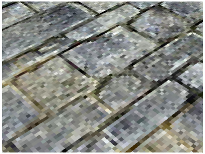
Naive interpolation leads to incorrectly rendered image:

Solution — perspective-correct interpolation:



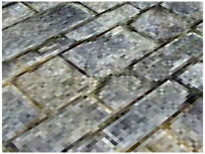
Pixel Processing

Texture magnification:



Pixel Processing

Texture magnification:



Solution — bilinear filtering:

Pixel Processing

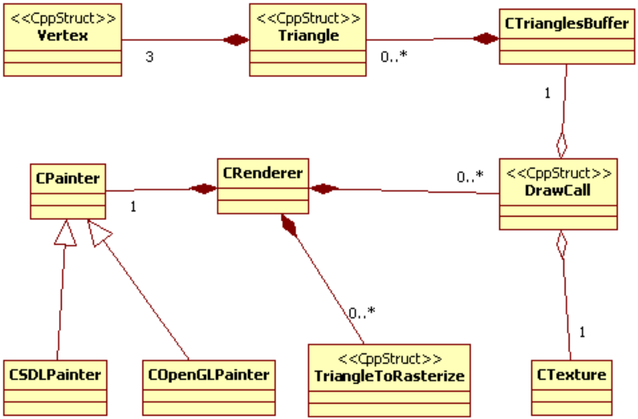
Texture minification:

Pixel Processing

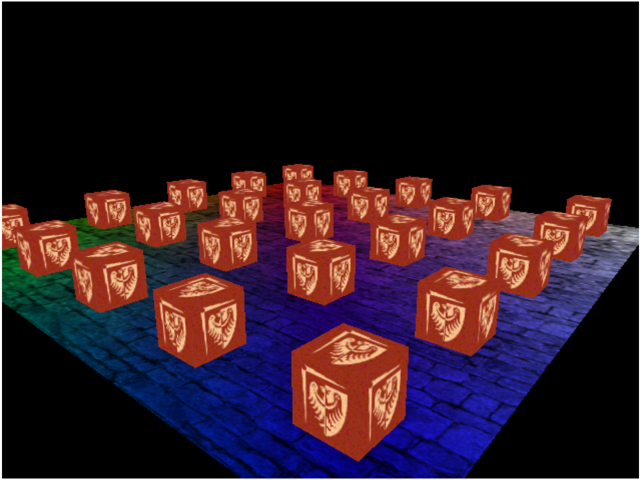
Texture minification:

Solution — mipmapping:

UML Class Diagram



Demo



Performance Tests

Performance Tests

Test 1 — a few of huge triangles:

Renderer	**Time**
Vainmoinen without CUDA	10400 ms
Vainmoinen with CUDA (One-Call-One-Triangle)	225 ms
Vainmoinen with CUDA (One-Call-Many-Triangles)	2200 ms
OpenGL	32 ms

Performance Tests

Test 1 — a few of huge triangles:

Renderer	Time
Vainmoinen without CUDA	10400 ms
Vainmoinen with CUDA (One-Call-One-Triangle)	225 ms
Vainmoinen with CUDA (One-Call-Many-Triangles)	2200 ms
OpenGL	32 ms

Test 2 — a lot of tiny triangles:

Renderer	Time
Vainmoinen without CUDA	24 ms
Vainmoinen with CUDA (One-Call-One-Triangle)	415 ms
Vainmoinen with CUDA (One-Call-Many-Triangles)	33 ms
OpenGL	22 ms

Conclusions and Future Work

Conclusions and Future Work

- implementation of the renderer itself has proven **not to be** that difficult

Conclusions and Future Work

- implementation of the renderer itself has proven **not to be** that difficult
- gaining the mathematical background needed to know how to implement the renderer has proven **to be** difficult

Conclusions and Future Work

- implementation of the renderer itself has proven **not to be** that difficult
- gaining the mathematical background needed to know how to implement the renderer has proven **to be** difficult
  it was also a great fun by the way

Conclusions and Future Work

- implementation of the renderer itself has proven **not to be** that difficult
- gaining the mathematical background needed to know how to implement the renderer has proven **to be** difficult
  it was also a great fun by the way
- CUDA can greatly speed up calculations

Conclusions and Future Work

- implementation of the renderer itself has proven **not to be** that difficult
- gaining the mathematical background needed to know how to implement the renderer has proven **to be** difficult
  it was also a great fun by the way
- CUDA can greatly speed up calculations
- further development of Vainmoinen would simply involve implementation of additional features

Thank you for the attention!