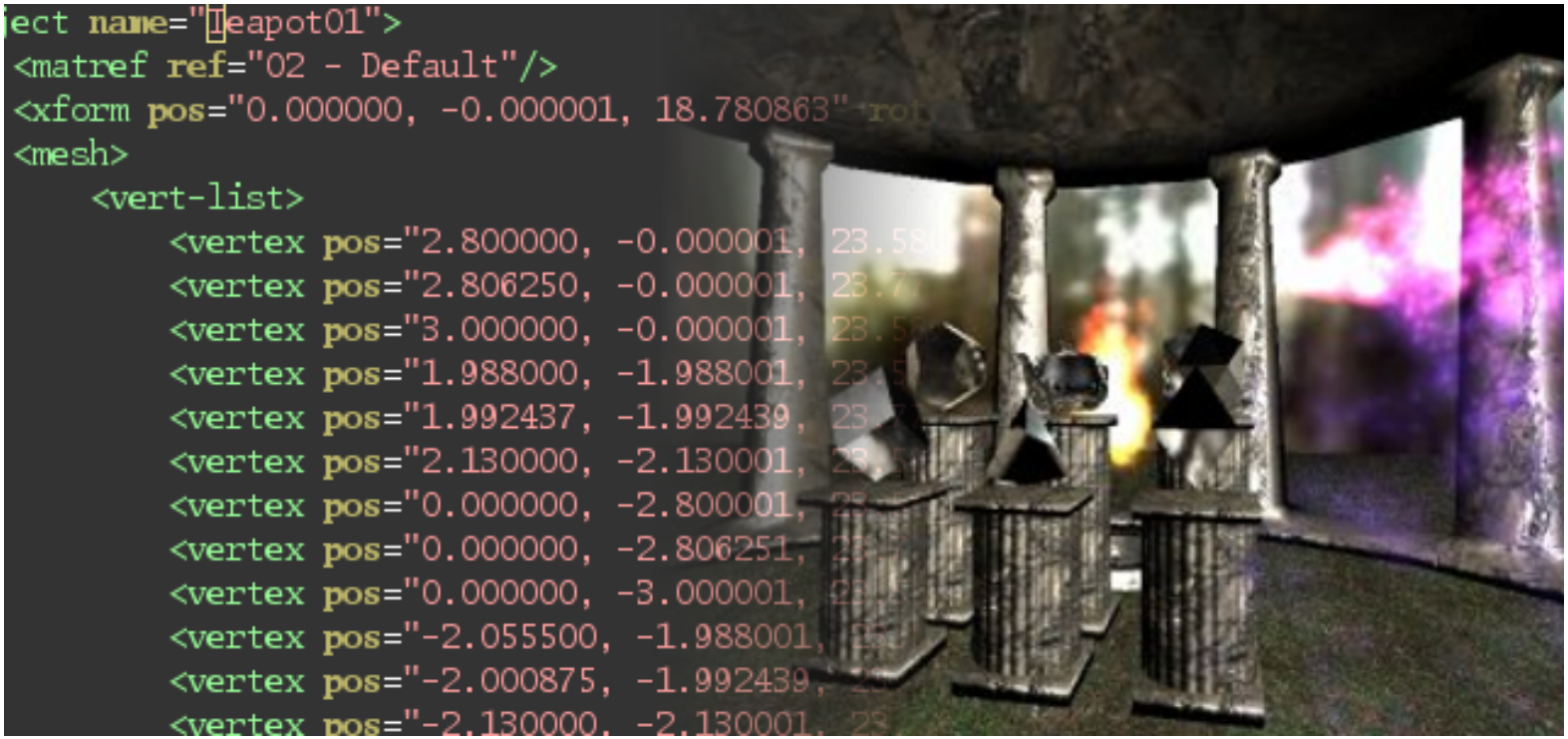
3dengfx: The Design and Implementation of 3D Engines

Ioannis Tsiombikas

December 21, 2007
Unipi free software community

Outline

In a nutshell: algorithms that transform a mathematical representation of a 3D environment, into a displayable image.



Typical Uses of 3D Graphics

- Entertainment
  - 3D animated movies (Toy Story, Ice Age, etc.)
  - Cinema special effects (Terminator, Star Wars, LoTR).
  - Computer games (well. . . all of them).
- Scientific Visualization (geological visualizations. fluid simulations, etc.)
- Medical Visualization (MRI, CT)
- Lately also user interfaces (beryl).

Graphics Algorithms Classification

The big tradeoff:

Realtime rendering

- Trades rendering quality for runtime efficiency.
- Usually based on *polygon filling* which is:
  - Fast!
  - Hackish.
  - Lots of hardware implementations.

Off-line rendering

- Trades runtime efficiency for rendering quality.
- Usually based on *ray tracing* which is:
  - Beautiful.
  - Elegant!
  - Intuitive...
  - Slow as hell (esp. with the addition of monte-carlo methods).

Graphics Algorithms Classification

The big tradeoff:

Realtime rendering

- Trades rendering quality for runtime efficiency.
- Usually based on *polygon filling* which is:
  - Fast!
  - Hackish.
  - Lots of hardware implementations.

Off-line rendering

- Trades runtime efficiency for rendering quality.
- Usually based on *ray tracing* which is:
  - Beautiful.
  - Elegant!
  - Intuitive...
  - Slow as hell (esp. with the addition of monte-carlo methods).

Graphics Algorithms Classification

The big tradeoff:

Realtime rendering

- Trades rendering quality for runtime efficiency.
- Usually based on *polygon filling* which is:
  - Fast!
  - Hackish.
  - Lots of hardware implementations.

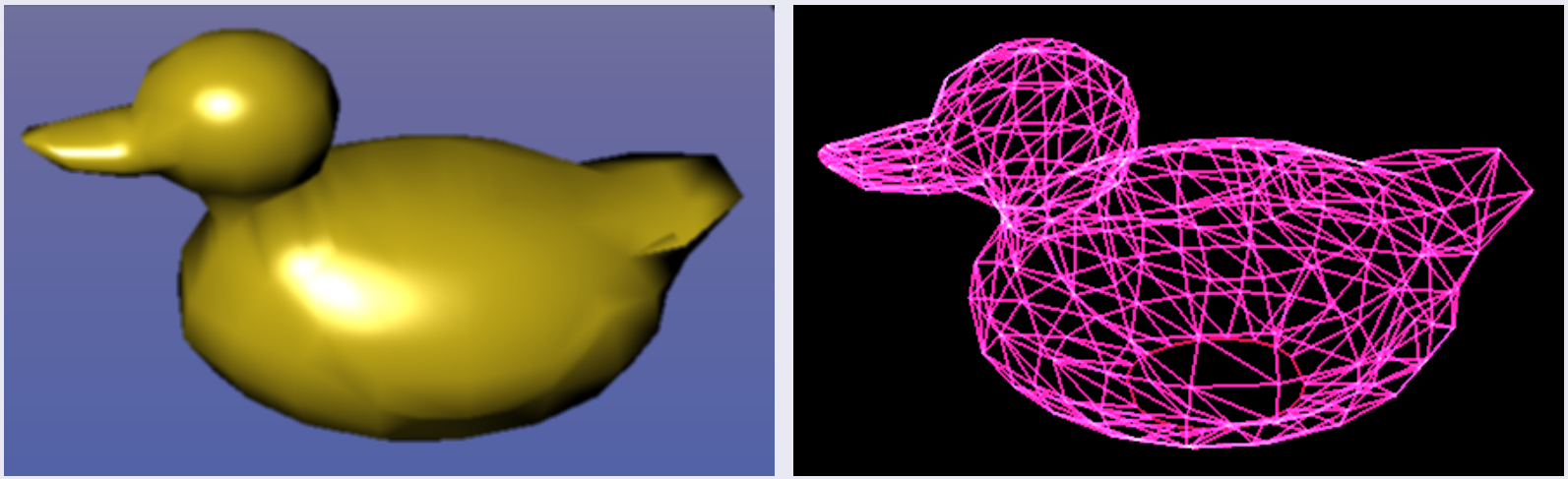
Off-line rendering

- Trades runtime efficiency for rendering quality.
- Usually based on *ray tracing* which is:
  - Beautiful.
  - Elegant!
  - Intuitive...
  - Slow as hell (esp. with the addition of monte-carlo methods).

Surface Representation

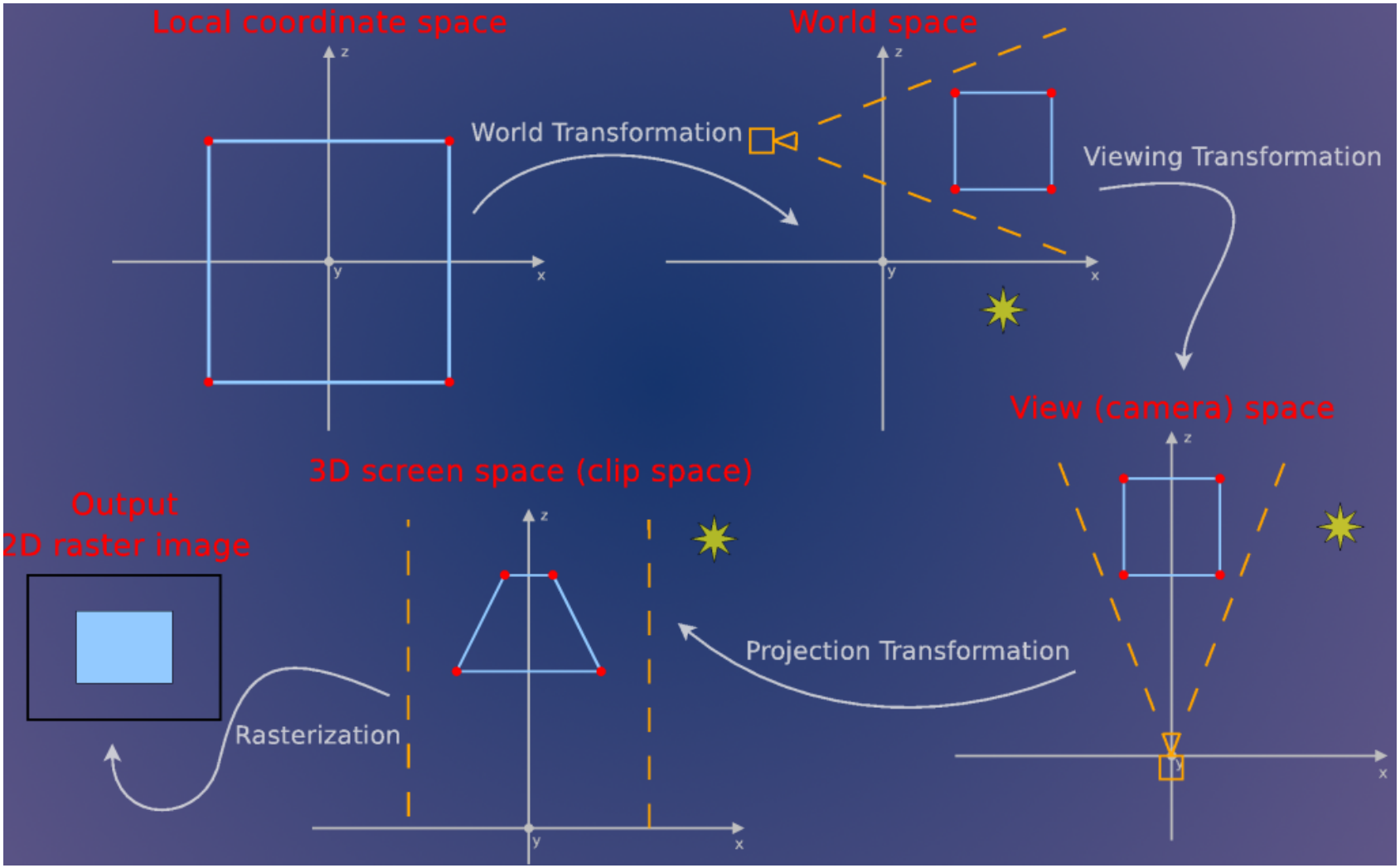
A 3D object is represented by a mesh of polygons, approximating the surface of the object.

A rubber duck



Coordinate Systems

During rendering, the vertices of each polygon are transformed through a number of coordinate systems.



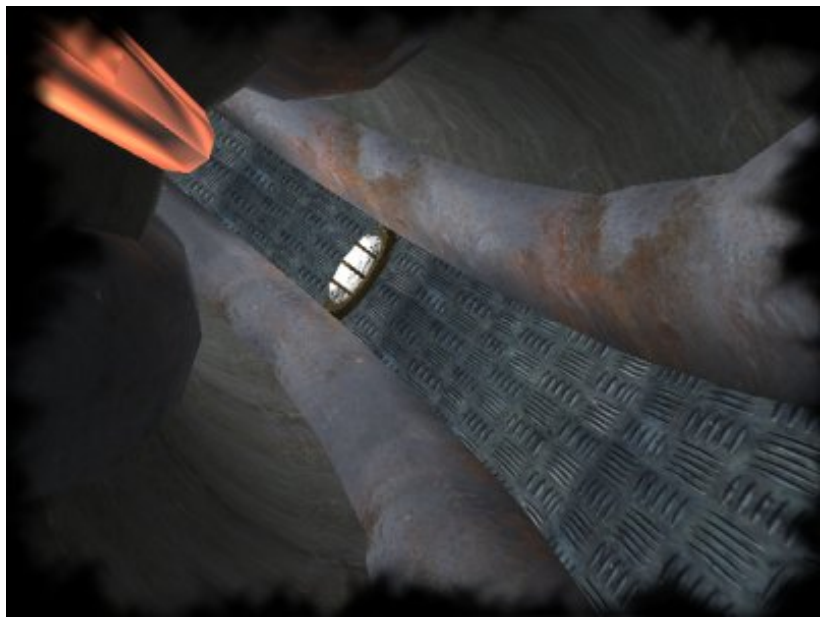
What is a 3D Engine?

A bunch of code, that handles various common 3D graphics tasks.
Usually a 3D engine handles most of the following:

- System abstraction.
- Low level drawing code / 3D graphics API handling.
- Scene database.
- Rendering algorithms.
- Special effects.
- Data import (models, textures, shaders, etc.)

About 3dengfx

3dengfx, is a 3D engine I started writing in mid-2003, mainly to be used in making "demos", and various graphics experiments.
During 2004, Michael Georgoulopoulos joined the project. Which continued until roughly the end of 2006.
Available (under the GPL) at: `http://engfx3d.berlios.de`.



3dengfx modules

The source directory structure, provides a quick overview of 3dengfx.

- 3dengfx (core).
- fxwt.
- gfx.
- n3dmath2.
- dsys.
- common.

3dengfx modules - 3dengfx

Includes anything that is specific to the 3D engine and can't be used separately.

- OpenGL low-level state manipulation and drawing.
- Virtual cameras, lights, shadows, and surface materials.
- Geometry generation, and loading (full scene loading from 3ds files).
- Texture and shader management.
- Abstraction of renderable "objects", with their state and rendering attributes.
- 3D scene management and automated full scene rendering.
- Particle systems.
- Volume rendering (marching cubes).

3dengfx modules - fxwt

Handles window system interaction and event handling, in a cross-platform manner.

- Creates OpenGL windows.
- Handles events (provides callback mechanism).
- Handles fonts / text rendering.

Compile-time selectable back-ends: Native(X11/GLX, Win32/WGL), SDL, GLUT, GTK+.

3dengfx modules - gfx

Anything graphics related, that isn't necessarily tied to 3dengfx or OpenGL.

- 3D geometry data structures (vertices, meshes, etc.)
- Animation (keyframe tracks, interpolation, hierarchy, controllers).
- Color related routines.
- Image loading / saving & image processing.

3dengfx modules - n3dmath2

This used to be a standalone library (and has become so again recently). Provides the fundamental (for computer graphics) mathematical primitives, and their operations.

- Vectors, cartesian and spherical.
- Matrices.
- Quaternions.
- Rays, quadratics and their intersections (not used for 3dengfx).
- Interpolation (linear, spline, bezier), numerical integration, gaussians, etc.

3dengfx modules - dsys

The "demosystem" part of 3dengfx.
A demo is broken into a sequence of "parts". Each part is added to the demosystem database.
A simple script interpreter handles the part sequencing.

Demoscript example

```
0 fx fade 0 2s <0,0,0,1> 0 <0,0,0,0> 0
0 fx overlay 1s 3s t0 sdr/radial_p.glsl

0 start_part partname
1000 set_rtarget partname t0
4000 set_rtarget partname fb
4000 stop_part partname
end
```

3dengfx modules - common

Common code, used by all other 3dengfx modules.

- Platform independent high-resolution timer.
- Configuration file parser.
- Byteorder determination, and I/O.
- Uniform error logging.
- Hash table, heapsort, fps counter, data file locator, etc.

3dengfx usage example

```
void display(void);
Scene *scene;

int main(void)
{
    create_graphics_context(800, 600, false);
    set_display_handler(display);
    scene = load_scene("foo.3ds");
    return main_loop();
}

void display(void)
{
    scene->render();
    flip();
}
```

What went right with 3dengfx

- Enabled very easy experimentations, quick&dirty prototypes, etc.
- Would have worked nicely for artist-heavy demos (or games). Got the last drops of juice out of 3ds files (animation, reflections, materials, hierarchies, etc).
- Great particle system.
- Ran perfectly on multiple operating systems (GNU/Linux, FreeBSD, IRIX, Windows), processor architectures (x86 32bit/64bit, MIPS 32bit/64bit).

What went wrong with 3dengfx

- Over-engineering. E.g. partial template specialization, just for providing a convenient "triangle array to index array" constructor.
- Too many abstractions and automation layers interacting in unforseen ways. Too many layers to peel in order to do something "less common".
- Eventually became too cumbersome to use for simple experiments, that wouldn't use all these abstractions anyway.

Theseis Engine

The theseis engine was designed and implemented, mostly by myself, Michael Georgoulopoulos, and Giorgos Markou.

Theseis Engine Design

Designed and implemented as a collection of separately built (but interdependent) libraries.

Some good parts from 3dengfx ripped and placed in the theseis engine: mathematical library, image loading/saving, and some other small pieces of code.

- Cross-platform and *cross-API*
- Modular, broken up into many specialized libraries.
- Heavily content-driven.
- Everything done exclusively with shaders.
- Implementing various cutting-edge rendering algorithms (deferred shading, realtime ambient occlusion, etc).

Theseis Engine Libraries

- libgc
- libgcx
- libsys
- math3d
- libcollision
- libppe
- libbadcfg
- libvfs
- libvis
- libimage
- libutk
- libchunk
- libcommon

Graphics Programmer's Toolbox (gfxtools)

New free software project! `http://gfxtools.sourceforge.net`

- A set of many, small, *independent*, extremely specialized libraries; covering everything generally needed by 3D graphics applications.
- Can be used separately as needed, or combined together, to provide most of the traditional notion of a 3D engine.
- Written in C, and keeping the API as simple as possible.

Benefits of the gfxtools Approach

- Most applications would use a very small part of 3dengfx. The rest was baggage, which this scheme eliminates.
- Very low learning overhead. People can easily pick up and use a small library, compared to a 30kloc catchall library like 3dengfx.
- Very low integration overhead. Each small piece can be easily integrated in a bigger program, to cover a specific need.
- C is supported correctly on virtually every platform. It's easier to `dlopen`/`dlsym`. And less prone to "provoke" overengineering tendencies.

Questions?

Thank you for listening!
Questions?