Frank de Jong

160010@buas.nl

Model files

In Plutonium

Table of Contents

[Motivation 2](#_Toc7346636)

[Premade work 3](#_Toc7346637)

[Geometry 3](#_Toc7346638)

[Effects 4](#_Toc7346639)

[Physics 5](#_Toc7346640)

# Motivation

Plutonium wants to be a physics focused engine with my components in the physics world, but most of the graphics will still come from models. For this we need an easy way to create models with physics objects attached to it.

As of now models will be used for general items, such as:

* Decoration objects
* Actors
* Buildings
* Environment

All of these models need to be able to contain the following global elements

* Geometry
  + Meshes
  + Materials
  + Mesh animations
  + Skeletal animations
* Effects
  + Particle emitters
  + Animated coats
* Physics
  + Rigid bodies
  + Cloth
  + Rope/chain
  + Fluid/Gas
  + Hair

All of these things need to be fast to load and not contains too much data that can be calculated. All of this is to create a highly responsive world that can be played through with minimal loading screens, allowing for continues gameplay.

The name for this model format will probably be Plutonium Model or .pum

# Premade work

Most of the elements in these formats have already been made, but because this is a learning project, I will not be using them in favor of slowly implementing them myself.

## Geometry

The geometry will store all static vertex information, static in this case meaning that it’s static to physics input. This will be used for objects that should not be affected by the physics world directly, this could include buildings, decorations, tree stumps and other solid and rigid objects.

It’s important here that the artist can work with preexisting tools that export to standardized formats. This is to allow fast integration of artists in the art pipeline and to allow multiple tools to be used to allow for a more fun workflow.

Currently multiple model formats have been looked at for this purpose:

|  |  |  |  |
| --- | --- | --- | --- |
| Format | Creator | Pros | Cons |
| GLTF | Khronos group | Standardized  Easy to read  Can be binary  Supports everything | Needs a lot of parsing  Requires extension support |
| OBJ | Wavefront | Standardized  Supports everything | No binary format  Large disk space required |
| M2 | Blizzard | Binary format  Supports everything  Fast to load | Internal format  Blizzard specific code |
| MD2 | Id Software | Binary format  Fast to load  Very small | Doesn’t support everything  Very restricting  Not supported |
| FBX | Autodesk | Standardized  Efficient format  Often used by artists  Supports everything | Changes often  Internal format |

## Effects

TODO

## Physics

I’m mainly looking at NVidia’s PhysX and VisualFX for the physics aspects of the models. The only real change I wish to make to this is to move the VisualFX physics into the global physics world so hair can react to changes in the world instead of just the local model space. The following Nvidia technologies are looked at:

* HairWorks
* Turf Effects
* WaveWorks
* Flow
* FlameWorks
* PhysX

The key to using physics will be to combine them in a global 3D grid space. With this only the required physics calculations are performed, and the quality of the physics can be modified to allow for realistic physics at close range whilst the lesser, far away physics object are affected less.

# Format requirements

The model needs to be fast to load and push to the GPU. For this we’ll make a similar separation of header and data as GLTF and M2 do. Although we want both of these to be in the same file to minimize disk lookups.

For this the file will be laid out in two sections, the first one will be the header and the second on being the data. Both of these sections will be saved in a binary format to speed up the loading process.

The header must contain at least the following information:

* Magic number (to prevent wrongful loading quickly)
* Version (to allow future parsers to handle older versions)
* Identifier (used to display information instead of the file name)
* Amounts (structure that stores the number of objects present)
  + Nodes
  + Meshes
  + Skins
  + Animations
  + Materials
  + Textures
  + Images
  + Samplers
  + Buffers
* Offsets (structure that stores the byte offsets to the objects)