PARSING QUAKE 3 GAME CONTENT

ID TECH 3

alias Quake 3 engine

- released in 1999
- open source since 2005
- written in C language

fixed function OpenGL era: Voodoo2, Geforce2

GAME ENGINE DESIGN

high level diagram

detailed documentation

- virtual machine with JIT support
- native: client, server, renderer
- VM: server game logic, client UI

IOQUAKE 3

improved and refactored version of Quake 3

souce code is on GitHub

GAME CONTENT

comes from PAK0.pk3 (zip file)

- game levels (.bsp)
- 3D models (.md3)
- textures (.tga, .jpg)
- visual materials (.shader)

BSP MAP

game level data (e.g. q3dm6.bsp)

- graphics geometry
- lightmaps
- references to materials
- collision geometry
- culling data (e.g. Binary Space Partition tree)

.bsp format description

USING BINARY

Efficient, pure binary (de)serialisation

using Data.Binary.Get

BSP loader source

GAME ENTITIES

stored in BSP file as a string

```
{
"origin" "192 -320 16"
"notfree" "1"
"classname" "weapon_shotgun"
}
{
"model" "*1"
"classname" "trigger_push"
"target" "t3"
}
{
"classname" "info_notnull"
"targetname" "t3"
"origin" "-1472 384 668"
}
```

weapons, armors, triggers (e.g. jump pad, teleport, door)

PARSER COMBINATORS

Parsec/Megaparsec/Attoparsec

using megaparsec

```
parseEntities fname src = case parse entities fname src of
  Left err  -> Left (parseErrorPretty err)
  Right e   -> Right e

entities :: Parser [EntityData]
entities = sc *> many entity <* eof

entity :: Parser EntityData
entity = foldr ($) defaultEntityData <$>
                  between (symbol "{") (symbol "}") (some value)

value :: Parser (EntityData -> EntityData)
value = stringLiteral >>= \case
  "classname"   -> (\v e -> e {classname = v}) <$> stringLiteral
```

Entity parser source

BUILTIN ITEM ENTITIES

Weapons, Ammos, Armors, Health

item descriptions in code

C source

Haskell source

MD3 MODEL

binary format for 3D models with animation

entity models (e.g. weapon)

player model

.md3 format description

MD3 loader source

MATERIAL

.shader files - declarative language to describe look, textures, colors, animations, geometry deformations

MATERIAL FORMAT

text based format for graphics artists

.shader file - declarative language

```
textures/skies/xtoxicsky_dm9
{
  skyparms full 512 -
  {
    map textures/skies/inteldimclouds.tga
    tcMod scroll 0.1 0.1
    tcMod scale 3 2
  }
  {
    map textures/skies/intelredclouds.tga
    blendFunc add
    tcMod scroll 0.05 0.05
    tcMod scale 3 3
  }
}
```

MATERIAL PARSER

no rendering, just store material description

data structure for materials: source

parser source using Attoparsec

RENDERING

render the geometry according the materials

LAMBDACUBE 3D

first create a rendering function for given materials

```
q3GFX :: [(String,CommonAttrs)] -> FB
```

then use the pure render function

- input: geometry, constants (time, camera)
- output: image

COLLISION DETECTION

ray trace against planes

collision brush = set of convex hulls defined by planes

1. find brushes in BSP tree
2. cull against brush planes

based on a blog post (C source)

rewritten in Haskell (source)

MAP VIEWER

- load .bsp level
- parse entities
- load .md3 models (for entities)
- parse materials
- generate render function for level materials
- load material textures
- pass textures and level geometry to render function
- main loop: update time and camera; render

SOURCE CODE

Check it out on GitHub:
https://github.com/lambdacube3d/lambdacube-quake3

QUESTIONS?