

# Adventures in Data Compilation

Uncharted: Drake's Fortune

Dan Liebgold

Naughty Dog, Inc.  
Santa Monica, CA

Game Developers Conference, 2008



# Motivation

- ▶ Code is compiled, data is “built”
- ▶ What should be code, what should be data? Plenty, right?
  - ▶ Game logic, geometry, textures...
- ▶ What is not clearly either?
  - ▶ Particle definitions, animation states & blend trees, event & gameplay scripting/tuning, more...



# Motivation

- ▶ Code is compiled, data is “built”
- ▶ What should be code, what should be data? Plenty, right?
  - ▶ Game logic, geometry, textures...
- ▶ What is not clearly either?
  - ▶ Particle definitions, animation states & blend trees, event & gameplay scripting/tuning, more...



# Motivation

- ▶ Code is compiled, data is “built”
- ▶ What should be code, what should be data? Plenty, right?
  - ▶ Game logic, geometry, textures...
- ▶ What is not clearly either?
  - ▶ Particle definitions, animation states & blend trees, event & gameplay scripting/tuning, more...



# Motivation

- ▶ Code is compiled, data is “built”
- ▶ What should be code, what should be data? Plenty, right?
  - ▶ Game logic, geometry, textures...
- ▶ What is not clearly either?
  - ▶ Particle definitions, animation states & blend trees, event & gameplay scripting/tuning, more...



# Motivation

- ▶ Code is compiled, data is “built”
- ▶ What should be code, what should be data? Plenty, right?
  - ▶ Game logic, geometry, textures...
- ▶ What is not clearly either?
  - ▶ Particle definitions, animation states & blend trees, event & gameplay scripting/tuning, more...



# The in between stuff

- ▶ Moving on from GOAL
- ▶ Lisp supports the code/data duality
- ▶ We will build DC in Scheme, a dialect of LISP!



# The in between stuff

- ▶ Moving on from GOAL
- ▶ Lisp supports the code/data duality
- ▶ We will build DC in Scheme, a dialect of LISP!

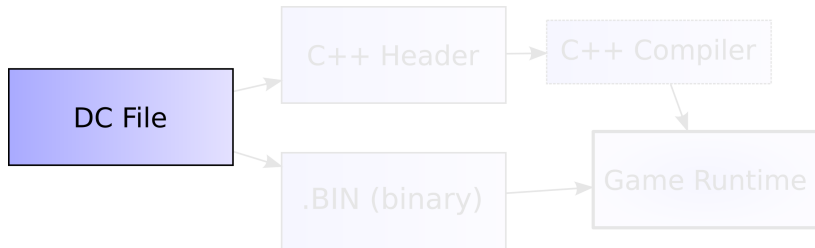


# The in between stuff

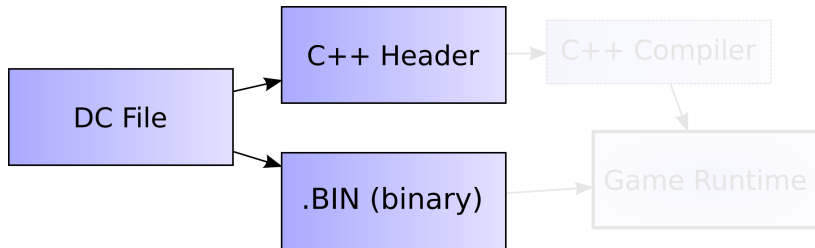
- ▶ Moving on from GOAL
- ▶ Lisp supports the code/data duality
- ▶ We will build DC in Scheme, a dialect of LISP!



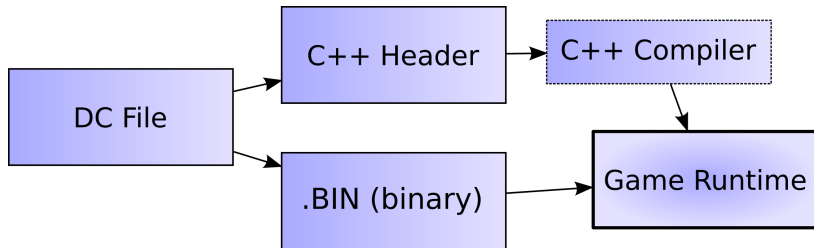
# Architecture



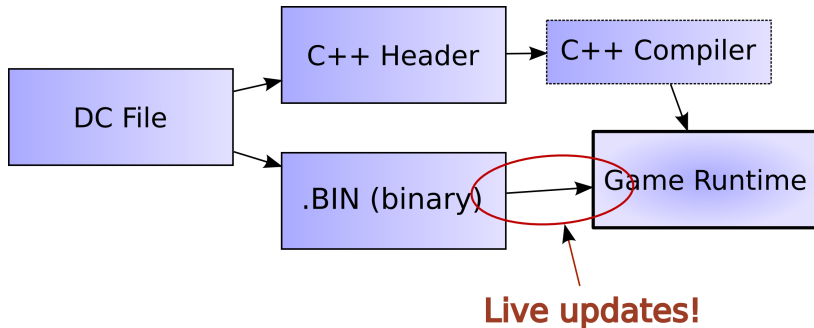
# Architecture



# Architecture



# Architecture



# Example

Let's define a player start position:

```
(define-export *player-start*  
  (new locator  
    :trans *origin*  
    :rot (axis-angle->quaternion *y-axis* 45)  
  ))
```

# Start with some types

```
(deftype vec4 (:align 16)
  ((x float)
   (y float)
   (z float)
   (w float :default 0)
  )
)
```

# Start with some types

```
(deftype vec4 (:align 16)
  (x float)
  (y float)
  (z float)
  (w float :default 0)
)
```

# Start with some types

```
(deftype vec4 (:align 16)
  ((x float)
   (y float)
   (z float)
   (w float :default 0)
  )
)
```

# Start with some types

```
(deftype vec4 (:align 16)
  ((x float)
   (y float)
   (z float)
   (w float :default 0)
  )
)
```

# Start with some types

```
(deftype vec4 (:align 16)
  ((x float)
   (y float)
   (z float)
   (w float :default 0)
  )
)
```

# Start with some types

```
(deftype vec4 (:align 16)
  ((x float)
   (y float)
   (z float)
   (w float :default 0)
  )
)
```

```
struct Vec4
{
  float m_x;
  float m_y;
  float m_z;
  float m_w;
};
```

## Types continued

```
(deftype quaternion (:parent vec4)
  ())
```

```
(deftype point (:parent vec4)
  ((w float :default 1)
   ))
```

```
(deftype locator ()
  ((trans point :inline #t)
   (rot quaternion :inline #t)
   )
 )
```

## Types continued

```
(deftype quaternion (:parent vec4)
  ())
```

```
(deftype point (:parent vec4)
  ((w float :default 1)
   ))
```

```
(deftype locator ()
  ((trans point :inline #t)
   (rot quaternion :inline #t)
   )
 )
```

## Types continued

```
(deftype quaternion (:parent vec4)
  ())
```

```
(deftype point (:parent vec4)
  ((w float :default 1)
   ))
```

```
struct Locator
{
  Point m_trans;
  Quaternion m_rot;
};
```

# Define a function

```
(define (axis-angle->quat axis angle)
  (let ((sin-angle/2 (sin (* 0.5 angle))))
    (new quaternion
      :x (* (-> axis x) sin-angle/2)
      :y (* (-> axis y) sin-angle/2)
      :z (* (-> axis z) sin-angle/2)
      :w (cos (* 0.5 angle))
    )))
```

## Define a function

```
(define (axis-angle->quat axis angle)
  (let ((sin-angle/2 (sin (* 0.5 angle))))
    (new quaternion
      :x (* (-> axis x) sin-angle/2)
      :y (* (-> axis y) sin-angle/2)
      :z (* (-> axis z) sin-angle/2)
      :w (cos (* 0.5 angle))
    )))
```

## Define a function

```
(define (axis-angle->quat axis angle)
  (let ((sin-angle/2 (sin (* 0.5 angle))))
    (new quaternion
      :x (* (-> axis x) sin-angle/2)
      :y (* (-> axis y) sin-angle/2)
      :z (* (-> axis z) sin-angle/2)
      :w (cos (* 0.5 angle))
    )))
```

## Define a function

```
(define (axis-angle->quat axis angle)
  (let ((sin-angle/2 (sin (* 0.5 angle))))
    (new quaternion
      :x (* (-> axis x) sin-angle/2)
      :y (* (-> axis y) sin-angle/2)
      :z (* (-> axis z) sin-angle/2)
      :w (cos (* 0.5 angle))
    )))
```

## Define some instances

```
(define *y-axis* (new vec4 :x 0 :y 1 :z 0))  
(define *origin* (new point :x 0 :y 0 :z 0))
```

```
(define-export *player-start*  
  (new locator  
    :trans *origin*  
    :rot (axis-angle->quaternion *y-axis* 45)  
  ))
```

## Define some instances

```
(define *y-axis* (new vec4 :x 0 :y 1 :z 0))  
(define *origin* (new point :x 0 :y 0 :z 0))
```

```
(define-export *player-start*  
  (new locator  
    :trans *origin*  
    :rot (axis-angle->quaternion *y-axis* 45)  
  ))
```

## How we use these definitions in C++ code

```
...  
#include "dc-types.h"  
...  
const Locator * pLoc =  
    DcLookupSymbol("*player-start*");  
Point pos = pLoc->m_trans;  
...
```

# Build upon this basis

We build upon this basis to create many many things

- ▶ Particle definitions
- ▶ Animation states
- ▶ Gameplay scripts
- ▶ Scripted in-game cinematics
- ▶ Weapons tuning
- ▶ Sound and voice setup
- ▶ Overall game sequencing and control
- ▶ ...and more