c.def
(pronounced SEE-def)

Macromedia® Flash™
animation language
Inspiration

Macromedia® Flash™ weaknesses:

- Tedious navigation of complex paths
- GUI Cumbersome for non-trivial movies
- No batch object creation and placement

Solution:

- programmatically compose Flash movies with c.def™
The Process

• The Goals
  – Design a flexible language
  – ... within time constraints
  – Create a Ferris wheel animation

• What Made Them Possible?
  – Working backwards
  – Working together
  – Regression testing, incremental testing
Architecture

c.def™ source
CDEF/Lexer.g
CDEF/Parser.g
CDEF/Walker.g

CDEF.java
CDEF.swf
Features I

- Non-linear 2D array (keyframes)

<table>
<thead>
<tr>
<th>Layer / Frame</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**naive:**

<table>
<thead>
<tr>
<th>Layer / Block</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**c.def:**

- start Frame
- Tween length
- Tween action
Features II

- Circles drawn utilizing Bezier curves
  Curved lines are constructed using a Quadratic Bezier curve. The curve is specified using three points - the current drawing position, an off-curve control point and an on-curve anchor point which defines the end-point of the curve.

- Polygons created with a sequence of Cartesian translations
Actions

: #("Document" name=id a=param b=param doc_body:.)
| #("Glyph" name=id a=param glyph_body:. )
| #("Path" name=id c=numericexp path_body:. )
| #(INSERT name=id a=param)
| #("Render" name=id (a=param | c=numericexp)
  (name2=id | b=param))
| #("Rotate" name=id c=numericexp a=param )
| #("SetColor" name=id a=param)
| #("for" name=id a=param for_body:. )
| #("if" c=numericexp if_body:. ( else_body:.))? 
| #(BODY (expr [parent] )* );
Convenient Notation

- Makes code easier to read – no comma-separated lists of numbers
- Colors
  - #Name ↔ 16 pre-defined colors
  - #(red, green, blue)
- Ranges
  - ->(from, to)
  - ->(from, to, step)
- Coordinates
  - &(x, y)
Drawing Statements

: "point"^ ... coords ...
| "line"^ ... coords ... coords ...
| "circle"^ ... coords ... expr ...
| "rect"^ ... coords ... coords ...
| "ellipse"^ ... coords ... expr ... expr ...
| "polygon"^ ... coords (COMMA! coords)* ...
| "color"^ ... color ...
| "fillcolor"^ ... color ...
| ID^ ... coords ...
Nitty Gritty

• Static Scoping

```java
if (parent instanceof CDEFDocument) {
    CDEFDocument doc = (CDEFDocument)parent;
    doc.setParams((CDEFXY)a, (CDEFColor)b);
    ipt.setDocument(doc);
    ipt.enterScope();
    expr(doc_body, doc);
    ipt.exitScope();
}
```
Demo

• (A quick and working one!)
Document d[&(400, 400), #White]
{
 /* Ferris wheel base */
 Glyph base[&(100, 0)]
{
    fillcolor[#(215, 155, 251)];
    polygon[&(100, 0), &(-100,170), &(0, 10),
            &(200, 0), &(0, -10), &(-100, -170)];
    fillcolor[#None];

    polygon[&(60, -170), &(20, 0), &(80, 10),
            &(0, 20)];
    circle[&(35, -155), 5];
    ...

}}
Nesting Glyphs, Rotation

/* A wheel's spike */
Glyph spike[&(0, 0)]
{
    line [&(20, 0), &(120, 0)];
    line [&(120, 0), &(58, 105)];
}

/* Define the wheel */
Glyph wheel[&(150, 150)]
{
    for[ i: ->( 1, 6 ) ]
    {
        Rotate [spike, 60, &(0, 0)];

        /* Place it onto the wheel */
        spike [&(150, 150)];
    }
}
Translation Over Paths

for[i : ->(1, 6) ]
{
  Path circularPath[i * (100/6)]
  { circle[&(200, 200), 120];
    point[&(0, 120)];
  }

  if[i % 2 == 0 ]
  {
    Render [ferrisCar, ->(1, 180), circularPath];
  }
  else
  {
    Render [ferrisCar2, ->(1, 180), circularPath];
  }
}
Lessons Learned

• Got OOP? We do.
• Work backwards: maintain your sanity.
• Start early, spend Rogaine $$ on beer.
• Set expectations early.
Group Members

- **Dennis Rakhamimov**, Group Leader
  FrontEnd, Backend
- **Eric Poirier**
  Documentation, Presentation, and Testing
- **Charles Catanach**
  Backend, and Testing
- **Tecuan Flores**
  Documentation, Presentation, and Testing