Lukes's RAT Tricks

Pipeline tricks for a RAT world

The Goal (E. Goldratt, 1992)

- The production of a CG image is a "process of ongoing improvement": from a guessed starting point to the image on film, the main constraint is often time.
- Given the iterative nature of the process, a constraint on time is just a way to express a constraint on iteration count
- Yet, iterations seem to contribute to nicer pictures more than "raw" time

A faster turnaround time

A number of points in this chain are natural candidates for optimization, as a faster turnaround easily means more iterations

Three of them are:

- Wasted iteration cycles
- Manual “touchup” interventions
- Unlucky guesses for parameter values

“Premature optimization is the root of all Evil”, *D. Knuth*

Common problems

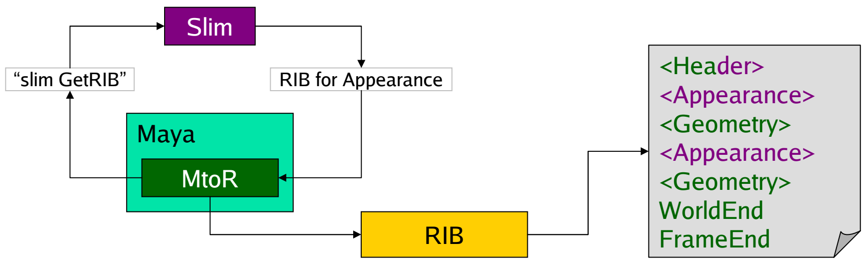
- *Waste*: often the startup procedure of an iteration is a nontrivial, error prone task
- *Touchups*: cycles are rarely completely machine handled, requiring some level of manual polish, e.g. to annotate results
- *Guessing*: the right value for a parameter can often be tricky to find, especially when visual orthogonality is not intrinsic to the environment

Ideas (a.k.a. them Tricks!)

- Extending the `slim` command: customizing MtoR's idea of what the look of a RIB file is
- Text in RenderMan shaders: most annotations are about parameters of the scene itself, better automate that!
- Fast iterations for bias and blur setup: an example of two highly non orthogonal settings, well known to all of us

Extending `slim`

An analysis of the RIB generation process in the MtoR/Slim marriage reveals that MtoR's idea of a RIB is heavily relying on Slim's perception of the world



Trick #1: `slim` subclassing

- The `slim` command can be thought as a singleton object: an object of a class that can be instantiated only once
- The goal of the trick is to instantiate the singleton from a subclass of the `slim` object itself, and make `MtoR` use our object as opposed to the original

TCL: `rename`

- The core of the trick is the TCL command `rename`: with this command you can change the name of any command or procedure in a TCL interpreter
- What we do is we build a procedure called `wrap` that renames the original command, and wraps it with two pieces of code coming from the user

TCL: wrap

```
proc wrap {func before after} {
  set newName [makeNewName $func]
  rename $func $newName
  proc $func {args} [subst -nocommands {
    $before
    set output [eval $newName \$args]
    $after
  }]
}
```

Wrapping slim

```
wrap slim {
   switch -exact [lindex $args 0] {
        getRIB {
                # overload logic here...
        }
   }
} {
   switch -exact [lindex $args 0] {
        getRIB {
                # more overload logic here,
                # possibly up to
                return $output
        }
   }
}
```

Results of Trick #1

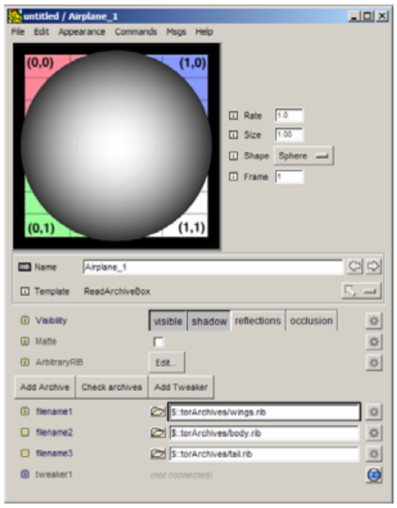
A first application of this trick is the injection of personalized data into the RIB stream in a studio/production dependent way

```
<Usual MtoR Header stuff here>
#slim subclassed _Frame_Setup
Option "user" "string context" [""]
Option "user" "string elementtype" ["final"]
Option "user" "string elementname" ["untitled"]
Option "user" "string jobname" ["untitled"]
Option "user" "string film" [“incredibles"]
Option "user" “float scene" [“12"]
Option "user" “float shot" [“5"]
WorldBegin
<...>
```

Results of Trick #1

Leveraging on that, is easy to create a RIBBox specialized in reading in RIB Archives only at the right time

```
IfBegin "$user:elementtype == 'final'
|| $user:elementtype == 'shadow'"
       ReadArchive "rib/wings.rib"
       ReadArchive "rib/body.rib"
       ReadArchive "rib/tail.rib"
IfEnd
```



Rate 1.0
Size 1.00
Shape Sphere
Frame 1
Name Airplane_1
Template ReadArchiveBox
Visibility [✓ visible] [✓ shadow] reflections occlusion
Matte ☐
ArbitraryRIB
filename1 $::torArchives/wings.rib
filename2 $::torArchives/body.rib
filename3 $::torArchives/tail.rib
tweaker1 (not connected)

Trick #2: text from a shader

A while ago Alex Segal wrote a DSO that would behave essentially like the `texture()` call in SL, but take EPS files as input

Such an approach is very convenient to use vector graphics in a render, as all text can conveniently be converted to outlines in most packages like Illustrator

This approach is well suited for logos and commercial graphics used in a render

Another step: digital typograhy

- A font is a collection of Beziér trim curves
- Hinting is what makes fonts legible
- PostScript curves don’t carry hints
- PostScript curves are unhandy to generate at render time from a font file (no, really, it’s hard!)

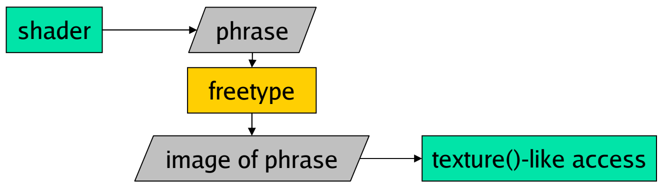
The last drop: I like `printf()`!

It would be useful to be able to render this piece of RIB:

```
Surface "text"
  "string fmt[]" ["Frame %d" "Scene %d"]
  "float argc[]" [1 1]
  "float argv[]" [13 123]
# Geometry: a square
```

May I present you... `freetype`!

- The freetype library is able to render character from over 10 families of formats of fonts
- It's by far the most used font rendering library around, at the base of the Qt/KDE and GTK+/Gnome projects

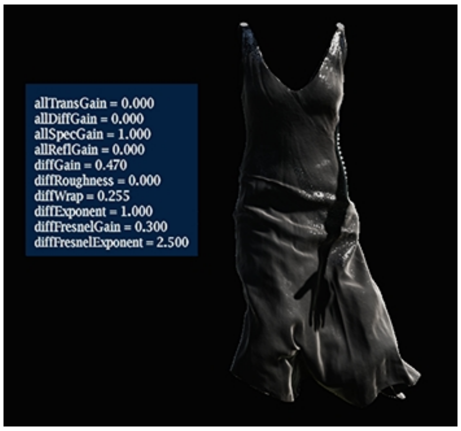


prmanText.so

```
surface note (float Kd = .7) {
 /* ... */
 uniform float _id = text_new ("diffuse = %.3g", 0,
   0, 1, 1);
 text_arg  (_id, Kd);
 text_font (_id, “/path/to/luxirr.ttf", 0, 100);
 float alpha = text(_id, s, t, ds, dt);
 Ci = Ci * (1 - alpha) + color (1,0,0) * alpha ;
 /* ... */
}
```

Results of Trick #2

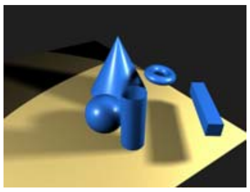
Annotating a wedge sequence is now an automatized task, providing for notes in your preview framebuffer, cooked onto your images

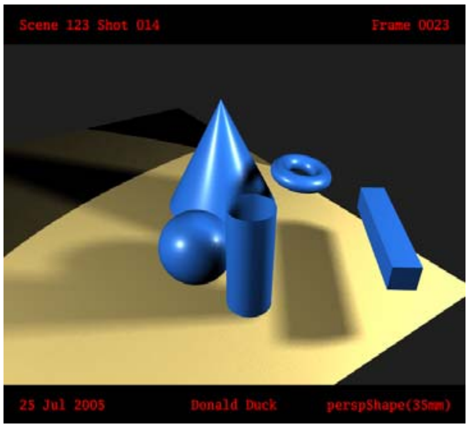


```
allTransGain = 0.000
allDiffGain = 0.000
allSpecGain = 1.000
allReflGain = 0.000
diffGain = 0.470
diffRoughness = 0.000
diffWrap = 0.255
diffExponent = 1.000
diffFresnelGain = 0.300
diffFresnelExponent = 2.500
```

Results of Trick #2

The frame decoration headers (usually added in comp) can now be accomplished through a wrapped `slim` command





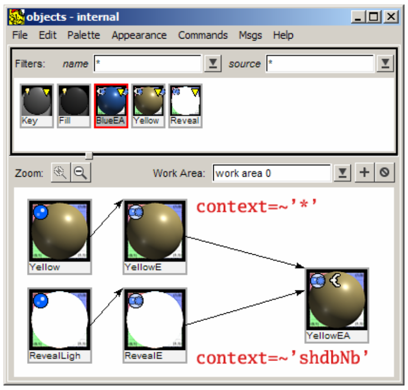
Trick #3: guessing bias & blur

- During lighting bias and blur setup often takes longer than actual artistic input
- Lighting intensity is affected by blurred shadow sampling as well
- Single light contributions are difficult to see in complex light rigs

A first step: shadowmap lighting

A way to actually look at what's going on is to use a simplified lighting model with one light at a time

```
surface shdbNb () {
  Ci = 0;
  illuminance (P) {
    Ci += Cl;
  }
}
```

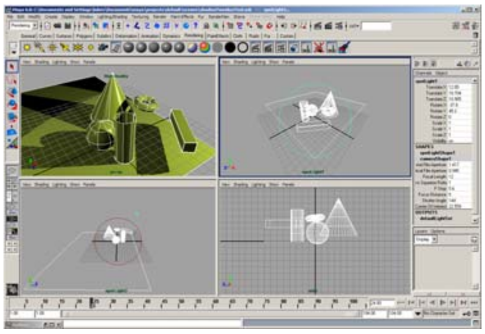


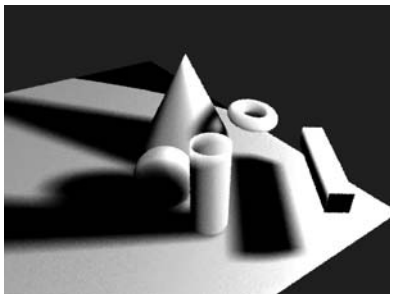
context=~'*'

context=~'shdbNb'

Shadowmap lighting

The highly contrasted image is an easier way to estimate the actual effect of the bias and blur settings

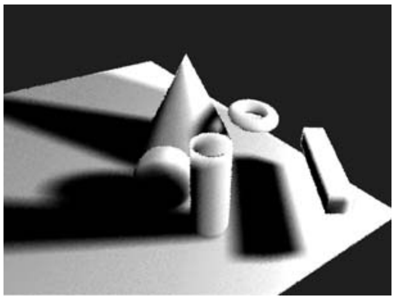




Just use shadow( ) then!

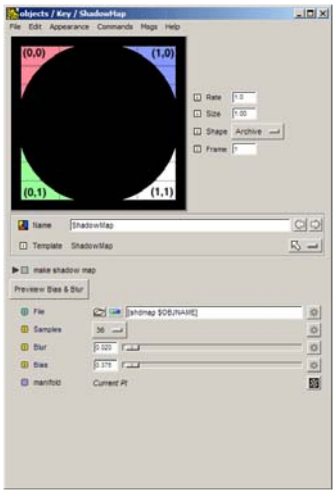
1. Cache geometry rendering P to a file
2. Put a plane in front of the camera
3. Use the cache as a texture to feed into shadow()

```
surface shdbNb2 (string pos =
“”; string shdmap = “”) {
  point myP = texture(pos);
  Ci = 1 - shadow(shdmap,myP);
}
```



Results of Trick #3

- The shadowmap render can be hooked into Slim’s shadowmap node
- Low rendertime (<5 s)
- Low memory usage, render can be local (no latency for startup)
- The RIB is fed from Slim into PRMan using TCL’s open `“|prman” w`
- Alfred is not needed


objects / Key / ShadowMap
File Edit Appearance Commands Maps Help
(0,0)
(1,0)
(0,1)
(1,1)
make shadow map
Preview Bias & Blur

Rate 1.0
Size 1.00
Shape Archive
Frame 1
Name ShadowMap
Template ShadowMap
File [shdmap $OBJNAME]
Samples 36
Blur 0.020
Bias 0.375
manifold Current Pt

The end

All the code will appear soon on
http://www.lucafascione.com

Write to me at
siggraph@lucafascione.com