Frame Buffer Object

Bedřich Beneš, Ph.D.
Purdue University
Department of Computer Graphics

PURDUE UNIVERSITY

Lecture Overview

- Arrays
- Multiple Render Target
- Frame Buffer Objects
- Case study:
  Motion blur using FBO

PURDUE UNIVERSITY

Arrays

- Arrays on the CPU
  are in the main memory

```
float a[200][200]; //static

float *b;
b=new float[1024]; //dynamic
```

etc…

PURDUE UNIVERSITY

Arrays

- Arrays on the GPU are textures

```
glGenTextures(1,&tId);//texture id
glBindTexture(GL_TEXTURE_2D,tId); //binding
glTexParameterf(GL_TEXTURE_2D,
   GL_TEXTURE_WRAP_S,GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D,
   GL_TEXTURE_WRAP_T,GL_REPEAT);
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB,
             width, height, 0, GL_RGB,
     GL_UNSIGNED_BYTE,0);//assigned 0 memory
```

Why do we want GPU arrays?

- Store data over multiple frames to minimize CPU-GPU communication
  - Less data transferred over PCIe
  - Reduce dependency on slow computer RAM.
- Increasing Speed!

Texture Target

OpenGL has two different 2D texture targets

`GL_TEXTURE_2D`

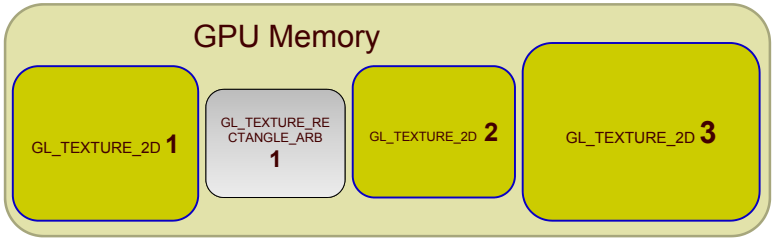
uses normalized texture coordinates <0,1>

`GL_TEXTURE_RECTANGLE`

uses normalized texture coordinates <0,1>
AND
integer texture coordinates (0,1,…,width-1)

GPU arrays concept

- `glGenTextures(1,&tId);//texture id`
  - Assigning a “pointer” or “identifier” to a block of GPU memory



GPU arrays concept

- OpenGL and Cg then use these identifiers to retrieve data from its own memory
- `glBindTexture(GL_TEXTURE_2D, ID)`
- `cgGLSetTextureParameter(param , ID)`
- `cgGLEnableTextureParameter(param)`

Reading from GPU arrays

Reading from textures/arrays is easy…

```
Uniform sampler2D myTexture;

vpEntry( … usual parameters …
          iTexCoord : TEXCOORD0 )
{
   float4 texColor=tex2D(myTexture,iTexCoord);
…
}
```


Writing to GPU arrays

- Writing is more difficult
  - There is no tex2DWrite() function
- We need to pass output from a shader into a texture somehow…

Motivation for the FBOs

- The “normal” rendering target is the frame buffer (FB)
- FBO is rendering into something “else”
- FBO is a container of (multiple) renderable object(s) (MRT)
- The main usage is
  a) off-screen rendering
  b) GPGPU ping-pong

FBO rendering Targets

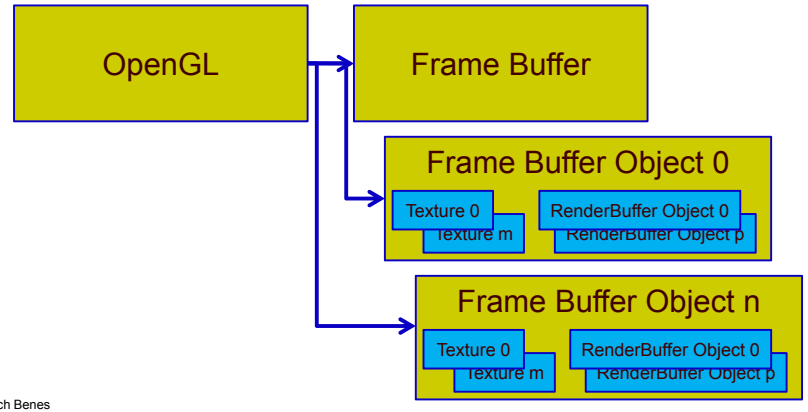
- Terminology:
  - Render to texture
    Image of a texture in FBO is the target
  - Off-screen Rendering
    Image of a renderbuffer in FBO is the target

FBO

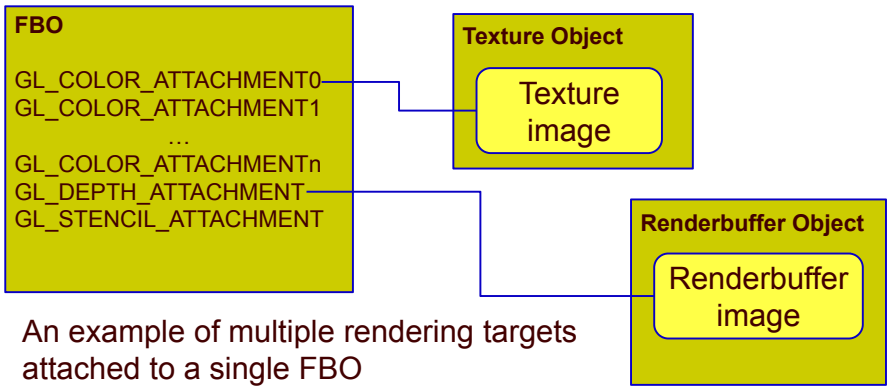
- The default FB is created by the windowing system
- FBO is application created (by you)
- FBO Supports: color, depth, and stencil
- FBO can use floating point values (!)

Motivation

- FBO Extension Overview



FBO



An example of multiple rendering targets attached to a single FBO
A texture contains image that is assigned to the FBO and an off-screen buffer stores a depth map

Multiple Render Targets (MRT)

- You can render into one or *more* renderable objects
- MRT is a rendering into more objects at the same time

Multiple Render Targets (MRT)

- Check how many buffers can be rendered at the same time

```
int maxbuffers;
glGetIntegerv(GL_MAX_COLOR_ATTACHMENTS, &maxbuffers);
```

- Attach them

```
GLenum mrt[]={GL_COLOR_ATTACHMENT0_EXT,
              GL_COLOR_ATTACHMENT1_EXT};
glDrawBuffers(2, mrt);
```


Writing to MRT in a shader

Use COLOR0, COLOR1 etc semantics

Example passthrough FP with MRT

```
void fpEntry(in  float4 iColor:COLOR0,
      in   float4 iColor1:COLOR1,
      out float4 oColor :COLOR0,
      out float4 oColor1:COLOR1)
{
   oColor = iColor;
   oColor1 = iColor1;
}
```


FBO setup for render to texture

1) Generate FBO
2) Create zero size texture
3) Bind it to the FBO
4) Attach the image to the FBO

FBO setup for render to texture

1) Generate FBO
2) Create zero size texture
3) Bind it to the FBO
4) Attach the image to the FBO

1) Generating a FBO

```
Gluint fboId;
glGenFramebuffersEXT(1, &fboId);
if (fboId==0) throw("NO FBO");
…
glDeleteFramebuffersEXT(1,&fboID);
```


2) Create zero size texture

```
glGenTextures(1, &textureId);
glBindTexture(GL_TEXTURE_2D, textureId);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
             width, height, 0, GL_RGB,
             GL_UNSIGNED_BYTE,0);
```


3) Bind the FBO

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,
                     fboId);
```

- Binding a FBO causes all buffer functions like glClear() glViewport() etc to ONLY effect that currently bound FBO

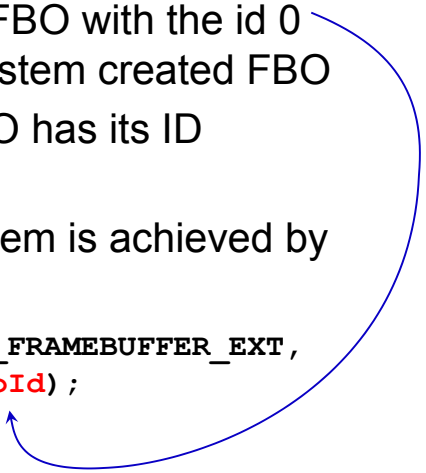
4) Attach 2D Texture to the FBO

```
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
   GL_COLOR_ATTACHMENT0_EXT,
   GL_TEXTURE_2D,textureId,0);
```

Enable Render to texture

- There is one default FBO with the id 0
  it is the windowing system created FBO
- The user-created FBO has its ID
  given by creation
- Switching between them is achieved by

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,
                     fboId);
```



Viewport

- Important detail!
  The rendering size of the FBO is given by the associated texture width and height
- It is usually different
  from the actual viewport size
- You can store/load the actual size by

```
glPushAttrib()/glPopAttrib()
```


Render to Texture

```
glPushAttrib(GL_VIEWPORT_BIT);
//set new viewport
glViewport(0,0,width,height);
//render to texture
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,
                     fboId);
//{rendering code goes here …}
//set rendering back to the actual screen
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
//restore the viewport
glPopAttrib();
```


FBO setup for off-screen render

1) Generate RBO
2) Bind it to the RBO
3) Assign storage to the RBO
4) Attach to the FBO

FBO setup for off-screen render

1) Generate RBO

2) Bind it to the RBO

3) Assign storage to the RBO

4) Attach to the FBO

1) Generating a RBO

```
Gluint rboId;
glGenRenderbuffersEXT(1, &rboId);
if (fboId==0) problem();
…
glDeleteRenderbuffersEXT(1,&rboID);
```


2) Bind the RBO

```
glBindRenderbuffersEXT(GL_RENDERBUFFER_EXT,
                       fboId);
```


3) Assign Storage

It does not have any data storage. Let's create it

```
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT,
   internalFormat, width, height);

internalFormat:
   GL_RGB, GL_RGBA, GL_DEPTH_COMPONENT, etc.
```

4) Attach the FBO

```
glFramebufferRenderbufferEXT(
   GL_FRAMEBUFFER_EXT,
   GL_DEPTH_ATTACHMENT_EXT,
   GL_RENDERBUFFER_EXT,rboId);
```


Cleanup

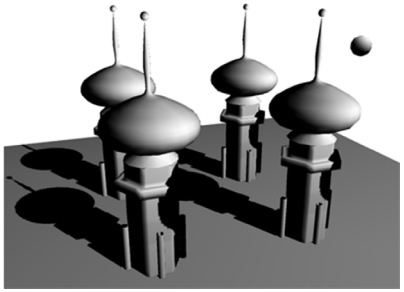
```
glDeleteTextures(1,&textureId);

//bind the default frame buffer
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

//delete FBO
glDeleteFramebuffersEXT(1, &fboId);
```

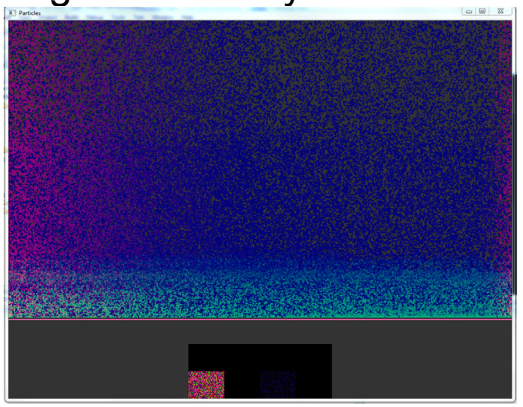

What are some of the uses of FBO?

- Shadow mapping
  - Render Depth to Texture



What are some of the uses of FBO?

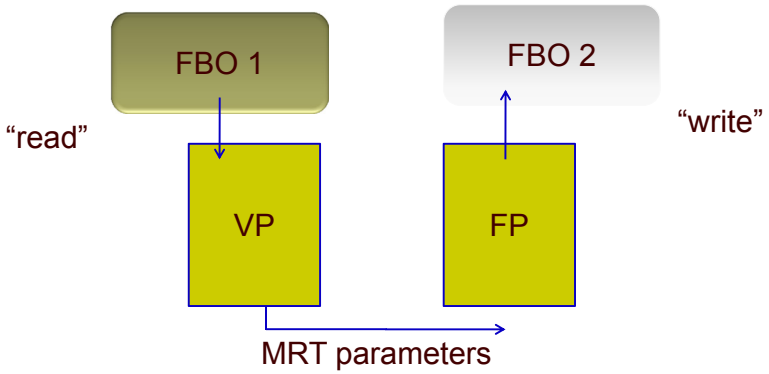
- GPGPU ping-pong : Particle System
- CPU limit 10,000
- GPU limit ?
  - Greater than 1mil…



GPGPU ping pong

- Creating a system that the GPU reads and writes its own memory without interruption from the CPU
- Requires double buffering of textures
  - Why? Cant read and write same texture at the same time

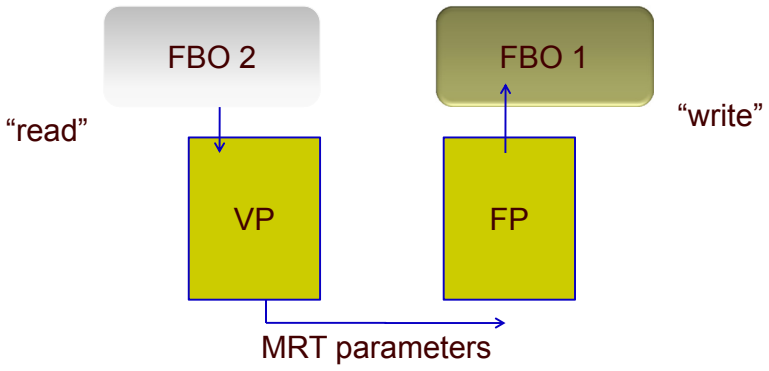
GPGPU ping pong layout



GPGPU ping pong layout

- No need to read back texture information each frame, we just 'switch' (ping-pong) the texture parameters using:
- `cgGLSetTextureParameter(cgreadparam , fbo1ID)`
- Then next frame…
- `cgGLSetTextureParameter(cgreadparam , fbo2ID)`

GPGPU ping pong layout #2



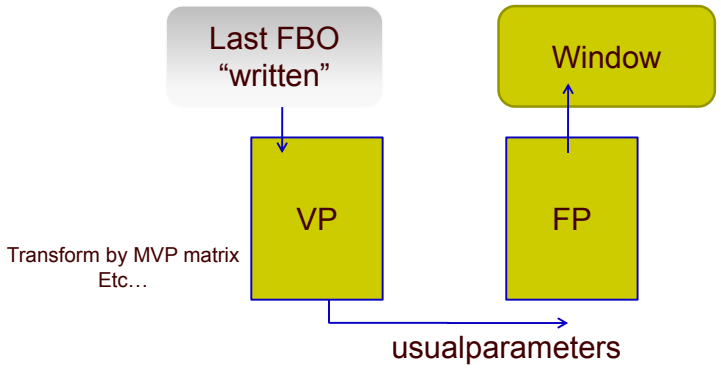
GPGPU Ping Pong

- Multi-pass approach
  - At least 2 passes
- Calculation Pass, render bound to FBO
- Beauty Pass, render bound to window

GPGPU Beauty pass

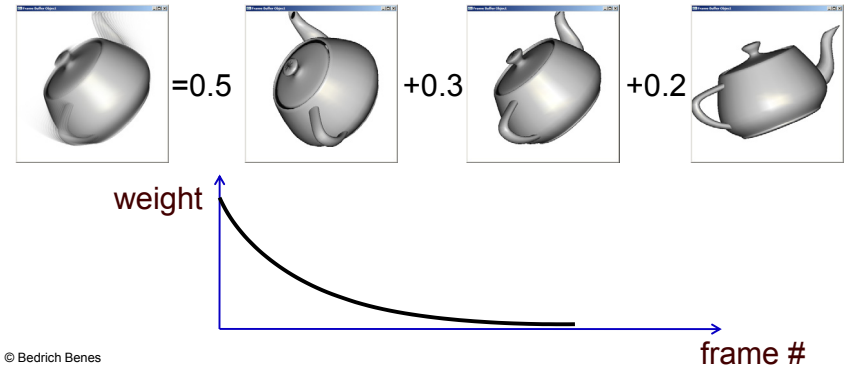
- Use the data stored in the textures, and actual camera information
  - Normal camera/rotation viewport etc…
- Transform the data into usable points in space
  - Render point sprites or teapots at the particle locations…

GPGPU Beauty pass



Motion Blur Using FBO

- Motion blur (time domain blur)



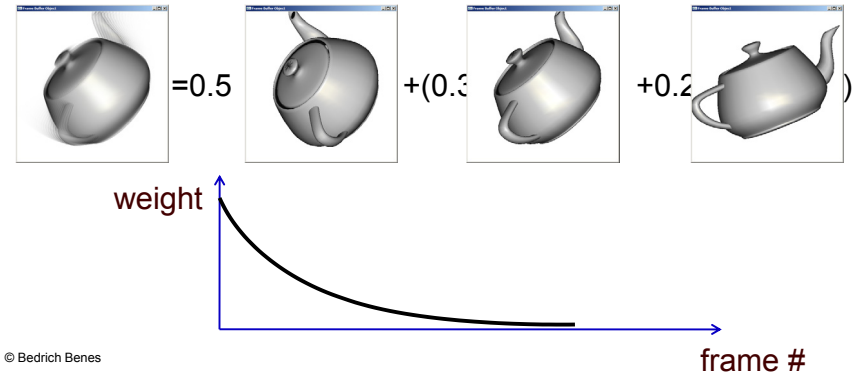
Motion Blur Using FBO

- Two textures – *screen* and *accum*
- Three passes

1st pass – render the scene to *screen*
2nd pass – diminish the *accum* and add the *screen* to it
3rd pass – copy the *accum* to frame buffer

Motion Blur Using FBO

- Motion blur (time domain blur)



We need one fragment shader

```
void accumBufferFP(
                in float2   iPos:WPOS, //fragment position
                uniform samplerRECT accum, //the two textures position
                uniform samplerRECT screen,
                uniform float       fade, //blurring coefficient
               out float4 oColor:COLOR) //output color
{
  oColor=(1-fade)*texRECT(screen,iPos)+fade*texRECT(accum,iPos);
}
```

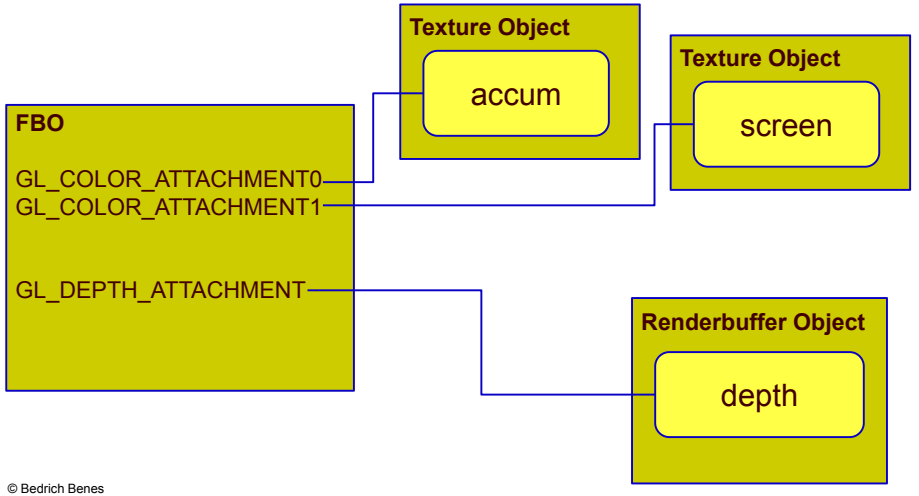

Initialization - OpenGL

```
glClearDepth(0xFFFFF);
glEnable(GL_DEPTH_TEST);
 glDepthFunc(GL_LEQUAL);//!!
glShadeModel(GL_SMOOTH);
glTexEnvf(GL_TEXTURE_ENV,
                GL_TEXTURE_ENV_MODE,
                GL_REPLACE);
```

Initialization - FBO

- One FBO with
  - Two textures *screen* and *accum*
  - One depth buffer
  - All objects are of fixed *width* x *height*

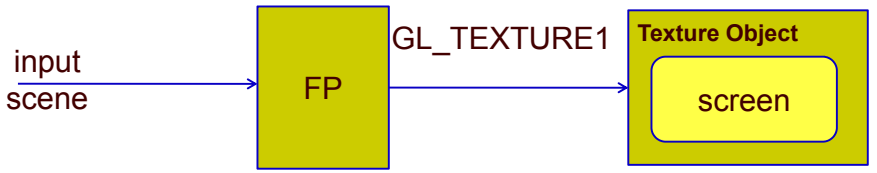
Initialization - FBO



1st pass: render 2 screen texture

```
glActiveTexture(GL_TEXTURE0);
glDisable(GL_TEXTURE_RECTANGLE_ARB);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,fbo); //Bind the FBO
glDrawBuffer(GL_COLOR_ATTACHMENT1_EXT); //render to texture
cgo.DisableFP(); //CG stuff
glViewport(0,0,fbo.GetWidth(),fbo.GetHeight());
//set transforms
//render the object
//Bind the Frame buffer
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```


1st pass: render 2 screen texture



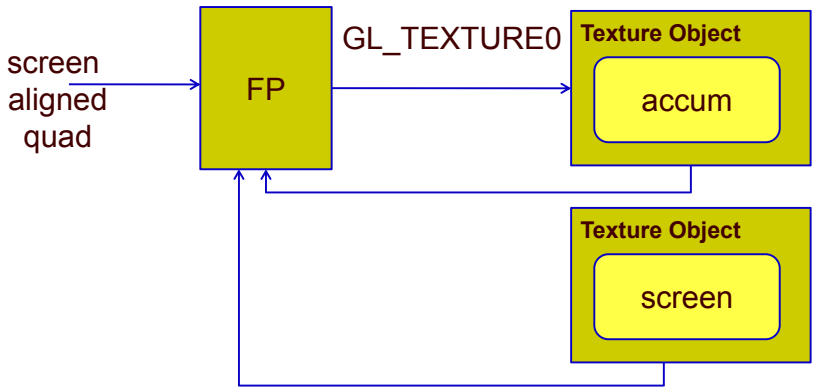
2nd pass: blend

- Render a screen aligned quad
- For each fragment on the screen
  - Read the *screen* texture texel
  - Read the *accum* texture texel
  - Blend them
  - Send the result to *accum*

2nd pass: blend

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,fbo); //Bind the FBO
glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT);//render to accum
//enable the shader and set texture parameters
cgo.EnableFPAccum();
cgGLSetTextureParameter(cgo.accum,fbo.GetTextureID());
cgGLSetTextureParameter(cgo.screen,fbo.GetScreenTextureID());
//send both textures
cgGLEnableTextureParameter(cgo.accum);
cgGLEnableTextureParameter(cgo.screen);
cgGLSetParameter1f(cgo.fade,cgo.fadef);
//Render the quad
//Bind the frame buffer
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```


2nd pass: blend



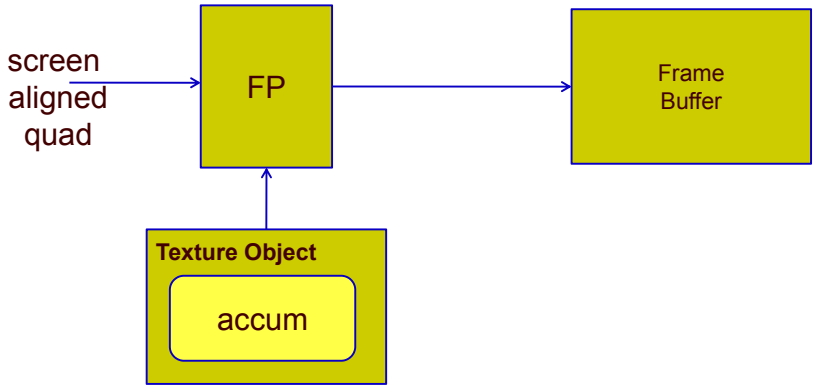
3rd Pass: copy *accum* to screen

- Render screen aligned quad
- For each fragment
  copy the corresponding
  accum texel on to output

3rd Pass: copy *accum* to screen

```
cgGLDisableProfile(fpProfile);//no shaders, use fixed pipeline
glDrawBuffer(GL_BACK); //Render to frame buffer glutSwap will do it
//bind the accumulation buffer
glBindTexture(GL_TEXTURE_RECTANGLE_EXT,textureID);
glEnable(GL_TEXTURE_RECTANGLE_EXT);//enable texturing
glBegin(GL_QUADS);
…
glEnd();
```


3rd Pass: copy *accum* to screen



Typical problems with FBOs

- Make sure where you render. glDrawBuffer()
- Make sure transformations are all right.
- Are you clearing the texture or the frame buffer? glClear();
- Are the right parameters attached to the Cg program?