ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

centre adscrit a: UNIVERSITAT DE BARCELONA

Computer Graphics

10. Transparency & Accumulation buffer

Dr Jesus Ojeda – jesusojeda@enti.cat

Contents

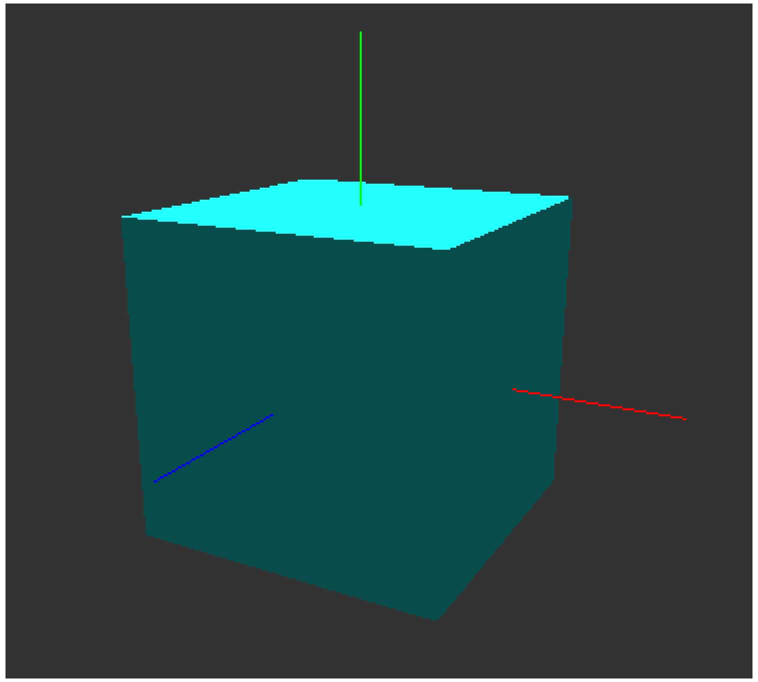
- Transparency
  - Polygon Stippling
  - Alpha blending
  - Caveats
- Accumulation buffer
  - How did it work
  - How it is implemented now
  - Advanced techniques

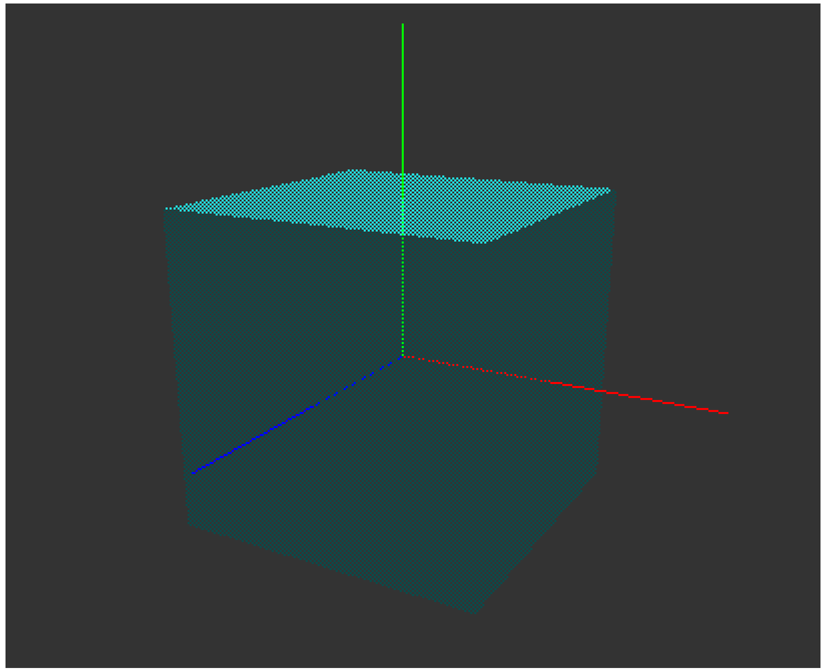
Transparency

- Until now all objects we have seen are opaque, they don’t let light pass through them.
- This allows the Z-buffer to early discard occluded geometry.
- What if we want to draw transparent / translucent objects?

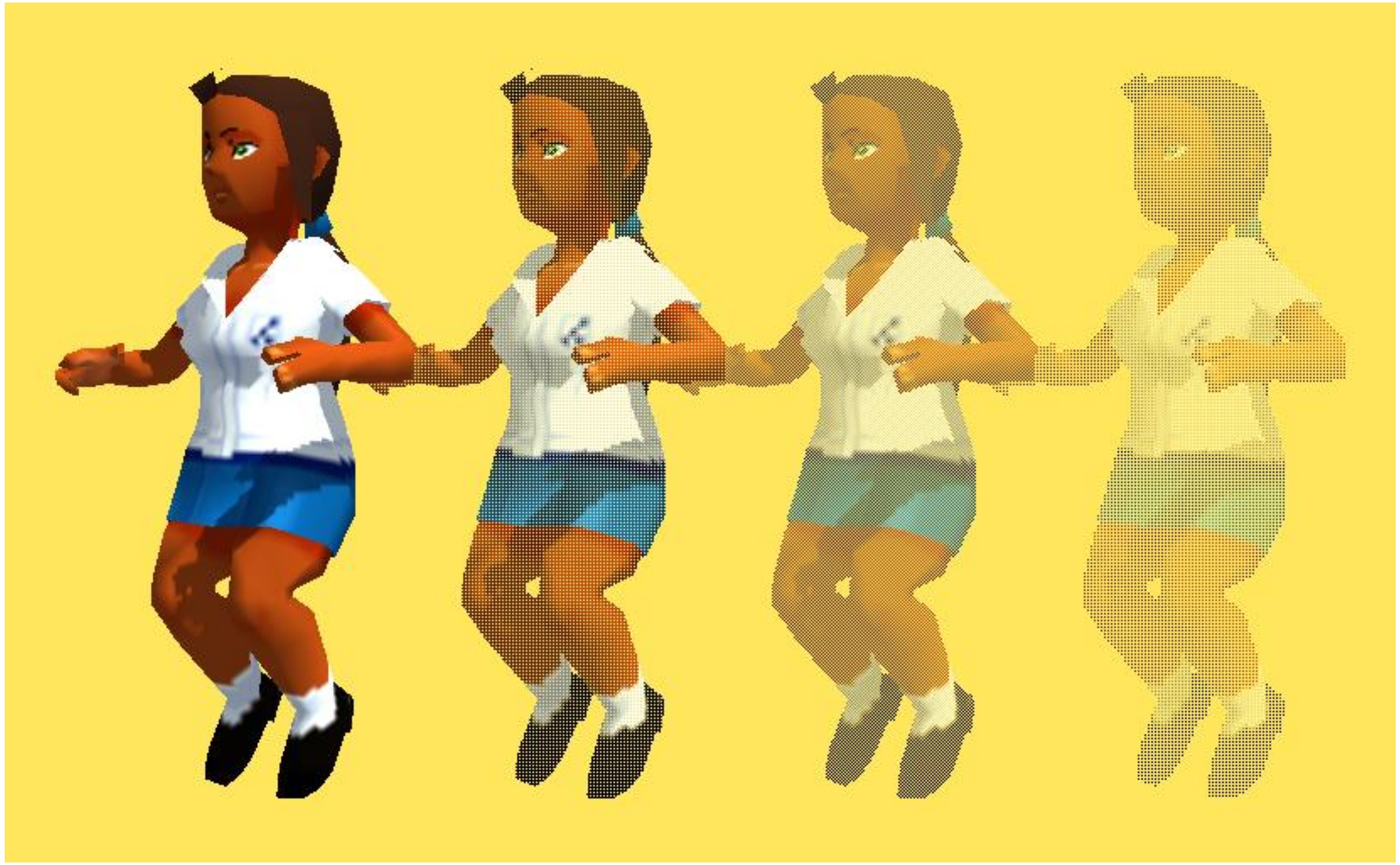
- Some solutions have to be devised for that.
- We will talk about polygon stippling and alpha blending.

Polygon stippling



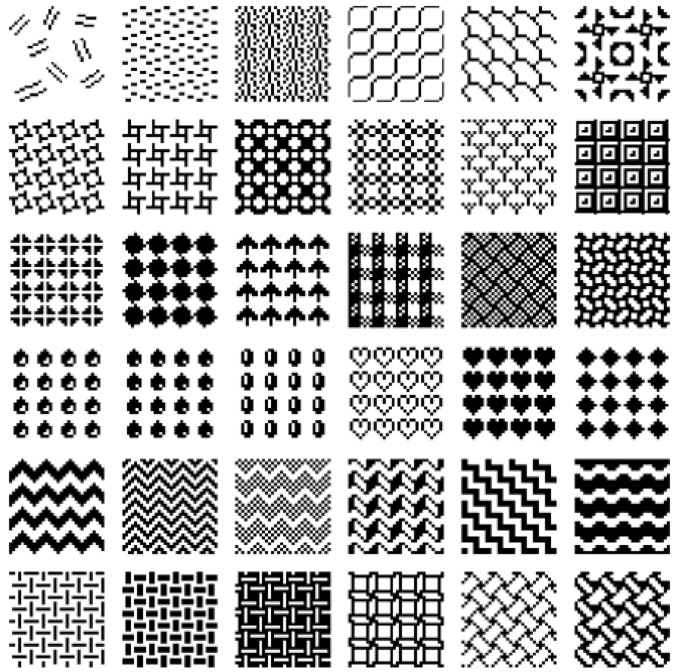


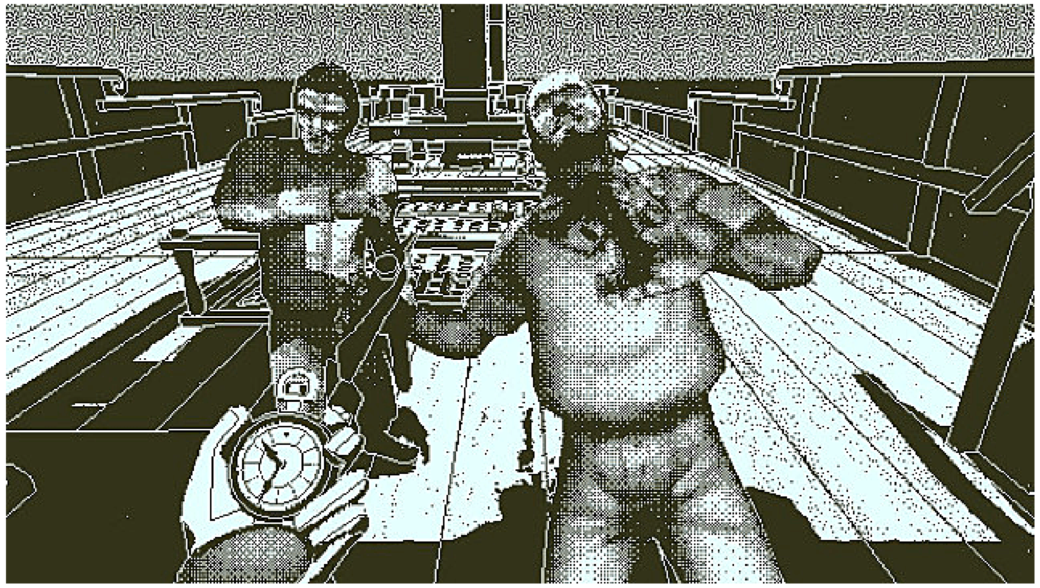
Polygon Stippling



Polygon Stippling

- Not only for transparency effects – but for artistic results.
- You could define multiple patterns for different effects.
- Implementation with the stencil buffer or in the fragment shader by discarding pixels on some condition.





Polygon Stippling – Let’s try the discard way

- In the simplest form, you will need to use the discard command from the fragment shader.

- As an exercise:
  - In a scene with multiple objects (cubes for example)
  - Make them discard a pixel if its window coordinates (gl_FragCoord) take some condition (whatever)
  - What happens if multiple objects use the same exact pattern?
  - Can you modify the pattern through time?

Alpha Blending

- What if we want semi-transparent objects?
- We control their opacity through alpha, but how are they composited?

Through a Blending Equation:

$\text{Color} = \text{Color}_{\text{source}} * \text{Factor}_{\text{source}} + \text{Color}_{\text{destination}} * \text{Factor}_{\text{destination}}$

Where:

- source is the writing color
- destination is the color already in the framebuffer

Alpha blending

Two things are needed:

1. glEnable(GL_BLEND)
2. glBlendFunc(
        GLenum *sfactor*,
        GLenum *dfactor*
)

Parameter	(fR,fG,fB,fA)fRfGfBfA
GL_ZERO	(0,0,0,0)0000
GL_ONE	(1,1,1,1)1111
GL_SRC_COLOR	(Rs0kR,Gs0kG,Bs0kB,As0kA)Rs0kRGs0kGBs0kBAs0kA
GL_ONE_MINUS_SRC_COLOR	(1,1,1,1)−(Rs0kR,Gs0kG,Bs0kB,As0kA)1111-Rs0kRGs0kGBs0kBAs0kA
GL_DST_COLOR	(RdkR,GdkG,BdkB,AdkA)RdkRGdkGBdkBAdkA
GL_ONE_MINUS_DST_COLOR	(1,1,1,1)−(RdkR,GdkG,BdkB,AdkA)1111-RdkRGdkGBdkBAdkA
GL_SRC_ALPHA	(As0kA,As0kA,As0kA,As0kA)As0kAAs0kAAs0kAAs0kA
GL_ONE_MINUS_SRC_ALPHA	(1,1,1,1)−(As0kA,As0kA,As0kA,As0kA)1111-As0kAAs0kAAs0kAAs0kA
GL_DST_ALPHA	(AdkA,AdkA,AdkA,AdkA)AdkAAdkAAdkAAdkA
GL_ONE_MINUS_DST_ALPHA	(1,1,1,1)−(AdkA,AdkA,AdkA,AdkA)1111-AdkAAdkAAdkAAdkA
GL_CONSTANT_COLOR	(Rc,Gc,Bc,Ac)RcGcBcAc
GL_ONE_MINUS_CONSTANT_COLOR	(1,1,1,1)−(Rc,Gc,Bc,Ac)1111-RcGcBcAc
GL_CONSTANT_ALPHA	(Ac,Ac,Ac,Ac)AcAcAcAc
GL_ONE_MINUS_CONSTANT_ALPHA	(1,1,1,1)−(Ac,Ac,Ac,Ac)1111-AcAcAcAc
GL_SRC_ALPHA_SATURATE	(i,i,i,1)iii1
GL_SRC1_COLOR	(Rs1kR,Gs1kG,Bs1kB,As1kA)Rs1kRGs1kGBs1kBAs1kA
GL_ONE_MINUS_SRC1_COLOR	(1,1,1,1)−(Rs1kR,Gs1kG,Bs1kB,As1kA)1111-Rs1kRGs1kGBs1kBAs1kA
GL_SRC1_ALPHA	(As1kA,As1kA,As1kA,As1kA)As1kAAs1kAAs1kAAs1kA
GL_ONE_MINUS_SRC1_ALPHA	(1,1,1,1)−(As1kA,As1kA,As1kA,As1kA)

Alpha blending

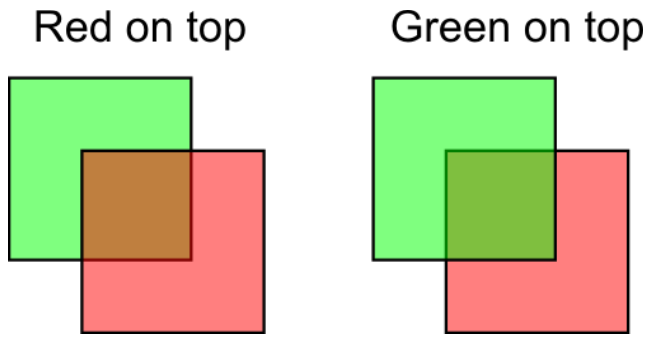
Let’s try it:

- Draw a cube with blending activated
- Draw a second cube with no blending
- Draw a third cube also with blending

- Try different order configurations
- Try different parameter combinations for the glBlendFunc function

Caveats

- With the Z-buffer enabled, we have relied on a correct rendering no matter the order of the objects on our draw commands.
- Not anymore, when blending order is important!



- The usual path: disable writing to Z-buffer and sort your scene (the meshes/triangles) in a back to front order. Render in that order.
- Alternatively: Order-Independent Transparency or Depth Peeling.

Accumulation buffer

- What if we want to create effects that combine in a blending multiple images in this frame?
- Or the image of the present frame with the image of previous frames?

- This blending would effectively *accumulate* color in each pixel of the framebuffer.

Accum buffer – The past

- Older versions of OpenGL had an accumulation buffer as an entity per se.
- If this accum buffer was active in the GL context you could clean it and write to it as to the traditional color buffer. With the addition of a multiplier as a blending factor.
- After some rendering (usually off-screen), the result on the accumulation buffer would be copied to the color buffer.

- For examples, take a look: https://www.glprogramming.com/red/chapter10.html#name3

Accum buffer – The present

- As we have now Framebuffer Objects to render off-screen, the newer versions of OpenGL have deprecated the Accumulation buffer.
- This means we can/must implement it on our own. Power to the people!

- How?
  - Render to multiple FBOs and blend them on a final pass, for example.

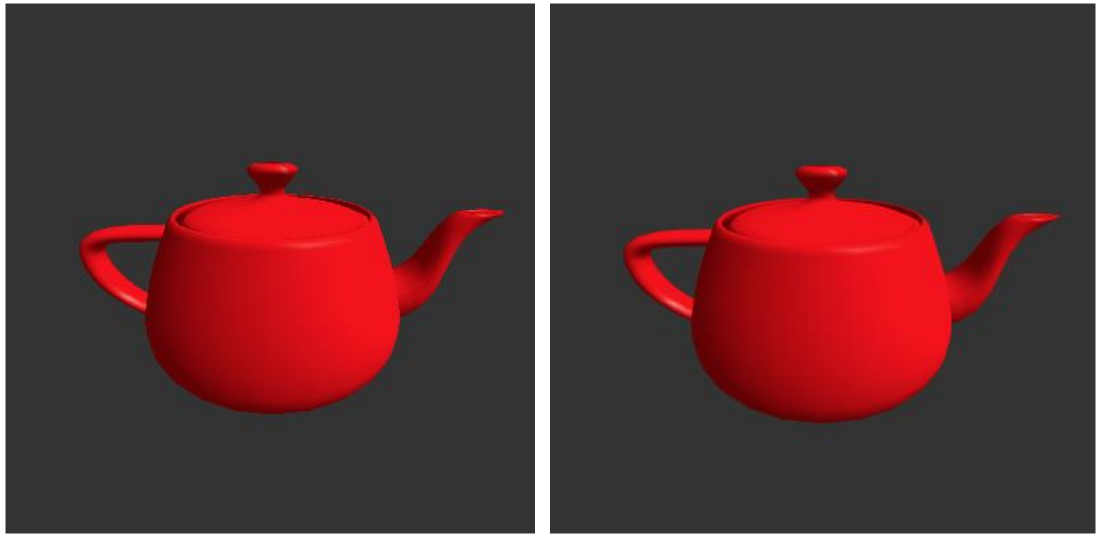
Accum buffer – Some advanced techniques

- Antialiasing
- Depth of field
- Motion Blur

Accum buffer – Antialiasing

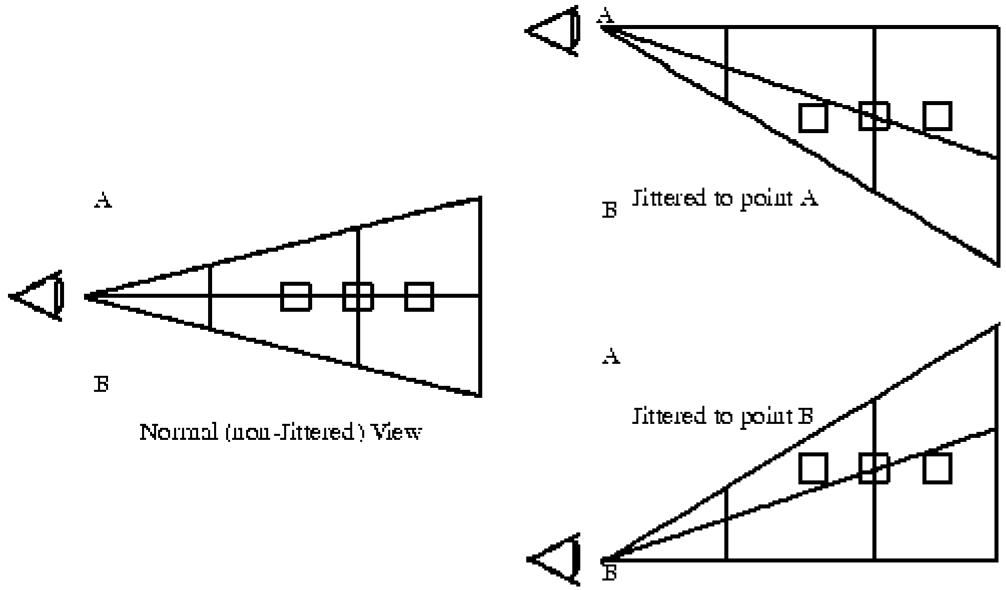
1. Render scene into buffer
2. Translate camera a subpixel amount in a perpendicular direction to the view direction
3. Render scene and accumulate/blend into buffer
4. Repeat for n times (in a jittered pattern around the initial camera position, for example)

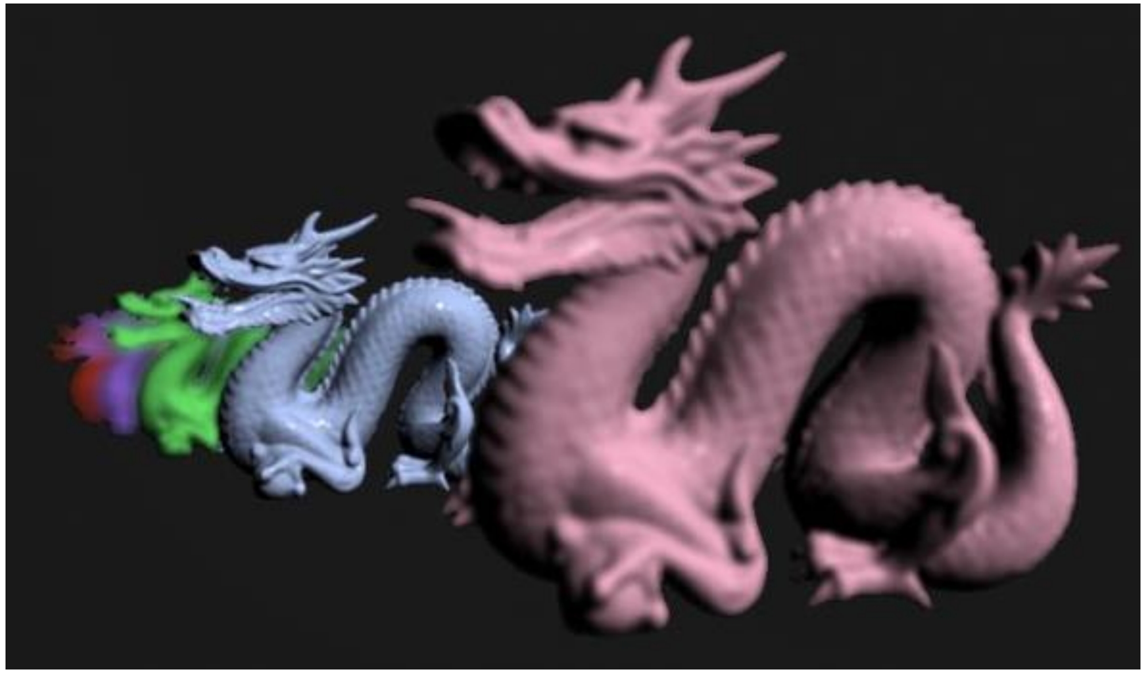
The blending is weighted =1, equally among each contributing image.



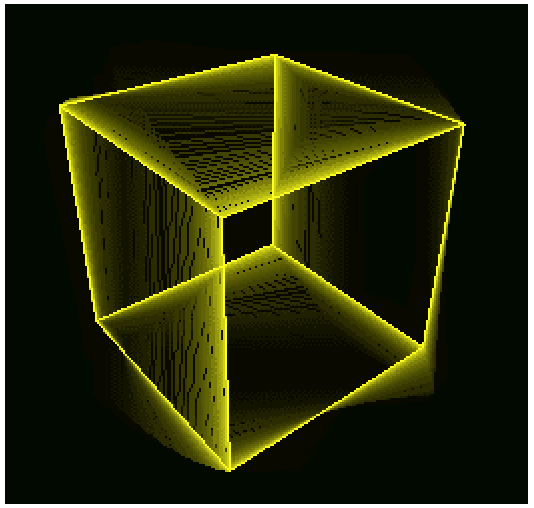
Accum buffer – Depth of Field

Same principle as in antialiasing but we will not only move the camera but change the projection frustum.

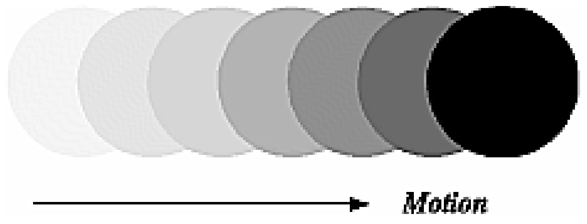




Accum buffer – Motion Blur



What about accumulating movement through time?
Motion Blur.



The same idea than before but without restricting the movement of the camera to subpixel translations. Now the objects are moved from time to time, accumulated in the framebuffer.

Also, to account for the fading of old "frames", the weights for the blending are selected so as to give more importance to late frames.

Resources

- Graham Sellers, Richard S. Writght, Jr. Nicholas Haemel. **OpenGL SuperBible**, 6th Edition. Pearson education.
- John Kessenich, Graham Sellers, Dave Shreiner. **OpenGL Programming guide**. Ninth Edition. Pearson Education.