Simple Motion Blur in OpenGL

Grundlangen Computergrafik

Hummel Felix
felix@agdsoft.com

Jenisch Alexander
ajenisch@cosy.sbg.ac.at

Simple Motion Blur in OpenGL

- **Overview,**
- **Motion blur techniques,**
- **The OpenGL accumulation buffer,**
- **Presentation,**
- **The end.**

Overview on motion blur

- Motion blur occurs:
  - in relative movement during exposure (e.g. motion picture),
  - during high latency of photo sensitive elements (e.g. human eye).
- Motion blur brings life to movement in:
  - games,
  - movies (animation),
  - still pictures.

Motion Blur Techniques

- **Trail modelling,**
- **Vertex shader,**
- **Accumulation buffer.**

Motion Blur Techniques: Trail modelling / Vertex shader

- The motion blur is modeled and rendered itself (e.g. a sword slicing through the air).

- Motion blur can also be done in two passes with vertex shaders:

  1. Pass:
     (a) The object is rendered normally.
  2. Pass:
     (a) Vertex shader applies previous and current frames' transform to each vertex.
     (b) The difference between these locations gives a motion vector per vertex.
     (c) If (motion vector)· (vertex's normal) is negative:
       - Then: Vertex's previous position is output.
       - Else: Vertex's current position is used.
     (d) Model streched out between set of polygons facing forward and backward.
     (e) Length of (motion vector): Alpha component of the backwardfacing vertices.

The OpenGL accumulation buffer

- **About,**
- **Specification,**
- **Constants,**
- **Example: Our implementation.**

The OpenGL accumulation buffer: About

- Extended-range color buffer:
  - Same screen resolution as color buffer,
  - Usually a higher bit depth.
- Each pixel consists of RGBA values,
- Images are **not** rendered into it,
- Possible effects: Antialiasing (points, lines, polygons), **motion blur**, depth of field,
- All operations are limitied to the area of the current *scissor box*.

The OpenGL accumulation buffer: Specification

To operate on the buffer, the use of **glAccum** is required.

C SPECIFICATION:

void **glAccum**(GLenum *op*, GLfloat *value*)

PARAMETERS:

op Specifies the accumulation buffer operation. Symbolic constans `GL_ACCUM`, `GL_LOAD`, `GL_ADD`, `GL_MULT`, and `GL_RETURN` are accepted.

value Specifies a floating-point value used in the accumulation buffer operation. *op* determines how *value* is used.

The OpenGL accumulation buffer: Constants

`GL_ACCUM` Obtains R, G, B, A values from buffer (selected for reading).
Each component divided by $2^n - 1$ (*n* bits allocated to each component).
Returns float $[0, 1]$, which is multiplied by *value*.
Outcome added to pixel component in the accumulation buffer.

`GL_LOAD` Similar to `GL_ACCUM`, only it overwrites the a. buffer.

`GL_MULT` Multipiles each R, G, B, A in the a. buffer by *value*.
Corresponding values stored in the a. buffer.

`GL_RETURN` Transfers values to the (color) buffer.
Multiplies R, G, B, A values **from** the a. b. by *value* and $2^n - 1$.
Values are clamped to the range $[0, 2^n - 1]$.
Values stored in corresponding display buffer cells.

The OpenGL accumulation buffer: Our implementation

Excerpt from *frame3D.cpp*:

```
void _draw(void)
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

frame->draw();

// q is a suited float (about .90 to .99)
glAccum(GL_MULT, q);
glAccum(GL_ACCUM, 1-q);
glAccum(GL_RETURN, 1.0);

glFlush();

glutSwapBuffers();
}
```

The OpenGL accumulation buffer: Our implementation (continued)

Excerpt from *frame3D.cpp*:

```
// function called when our frame is resized
void _resize(int w, int h)
{
    glClearAccum(0.0, 0.0, 0.0, 1.0);
    glClear(GL_ACCUM_BUFFER_BIT);
    frame->set_size(vector2D(w, h));
}
```

References

- **Real-Time Rendering, Second Edition, Tomas Akenine-Möller, Eric Haines, A K Peters 2002**

- **OpenGL Reference Manual, Second Edition, The Official Reference Document to OpenGL, Version 1.1, Renate Kempf, Chris Frazier, Addison Wesley 1998**

The End

After a short presentation of our project in action, you may ask questions and rejoice, you've made it through our presentation!