Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering

Steven P. Callahan, Milan Ikits, João L. D. Comba, and Cláudio T. Silva

APPENDIX I
VERTEX PROGRAM FOR HAVS

!!ARBfp1.0

# Vertex program for Transactions on Visualization and Computer Graphics;
# "Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering"
# (C) 2005 Steven Callahan, Milan Ikits, João Comba, Cláudio Silva
# ---------------------------------------------------------------

ATTRIB iPos = vertex.position;
ATTRIB iTex0 = vertex.texcoord[0];
PARAM mv[4] = \{ static.matrix.mvp[] \};
OUTPUT oPos = result.position;
OUTPUT oTex0 = result.texcoord[0];
OUTPUT oTex1 = result.texcoord[1];

# transform vertex to clip coordinates
DF4 oPos.x, mv[0], iPos;
DF4 oPos.y, mv[1], iPos;
DF4 oPos.z, mv[2], iPos;
DF4 oPos.w, mv[3], iPos;

# transform vertex to eye coordinates
DF4 oTex1.x, mv[0], iPos;
DF4 oTex1.y, mv[1], iPos;
DF4 oTex1.z, mv[2], iPos;

# texcoord 0 contains the scalar data value
HCOL oTex0, iTex0;
END

APPENDIX II
FRAGMENT PROGRAM FOR HAVS

!!ARBvp1.0

# Fragment program for Transactions on Visualization and Computer Graphics;
# "Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering"
# (C) 2005 Steven Callahan, Milan Ikits, João Comba, Cláudio Silva
# ---------------------------------------------------------------

The program consists of the following steps:
1. Find the first and second entries in the fixed size k-buffer list sorted by d (4+1 entries)
2. Perform a 3D pre-integrated transfer function lookup using front and back scalar data values and the segment length computed from the distance values of the first and second entries from the k-buffer.
3. Composite the color and opacity from the fragment buffer. Discard winning k-buffer entry, write the remaining k-buffer entries.

The following textures are used:
1. Tex 0: framebuffer (pbuffer, 2D RGBA 16/32 bpp float)
2. Tex 1: k-buffer entry 1 and 2 (same)
3. Tex 2: k-buffer entry 3 and 4 (same)
4. Tex 3: k-buffer entry 5 and 6 (same)
5. Tex 4: transfer function (regular, 3D RGBA 8/16 bpp int)

# use the ARB_draw_buffers extension
OPTION ARB_draw_buffers;
# this does not matter now, but will matter on future hardware
OPTION ARB_precision_hint_fastest;

# input and temporaries
ATTRIB p = fragment.position; # fragment position in screen space
ATTRIB v = fragment.texcoord[0]; # v.x = scalar value
PARAM sz = program.local[0]; # scale and bias parameters
# \{1/pwx, 1/ph, 1/z_size/max_len, 1/[2^w_size]\}
TEMP a0, a1, a2, a3, a4, a5, a6; # k-buffer entries
TEMP o0, o1, o2, o3, o4, o5, o6; # sorted results
TEMP v, o, p; # color and opacity
TEMP t; # temporary (boolean flag for min/max, dependent texture coordinate, # pbuffer texture coordinate, fragment to eye distance)

# compute texture coordinates from window position so that it is not # interpolated perspective correct. Then look up the color and opacity from # the framebuffer
HEX t, p, sz; # t.xy = p.xy * sz.xy, only x and y are used for texture lookup
TEX o0, t, texture[0], 2D; # k-buffer entry 0
TEX o1, t, texture[1], 2D; # k-buffer entry 1
TEX o2, t, texture[2], 2D; # k-buffer entry 2
TEX o3, t, texture[3], 2D; # k-buffer entry 3
TEX o4, t, texture[4], 2D; # k-buffer entry 4
TEX o5, t, texture[5], 2D; # k-buffer entry 5

# check opacity and kill fragment if it is greater than a constant tolerance
SUB o0, o0, o5; # kill fragment if o5 > o0

# find fragment with minimum d (r0), save the rest to r1, r2, r3, r4, r5
# r0 = min(r0, r1, r2, r3, r4, r5)
SUM o0, o5, o1; # t.xy = p.xy * sz.xy, only x and y are used for texture lookup

S. Callahan, M. Ikits, and C. Silva are with the Scientific Computing and Imaging Institute, University of Utah. J. Comba is with the Federal University of Rio Grande do Sul, Brazil.
TEX c, t, texture[4], xc; # look up pre-integrated color and opacity

# nullify winning entry if the scalar value < 0
CMP c, r0.x, 0.0, c;

# composite color with the color from the framebuffer !!!front to back!!!
SUB t.w, 1.0, r0.w;
MAD result.color[0], c, t.w, c;

# write remaining k-buffer entries
MOV r1.xw, r2.xxxy;
MOV r3.xw, r4.xxxy;
MOV r5.xw, r6.xxxy;
MOV result.color[1], r1;
MOV result.color[2], r3;
MOV result.color[3], r5;
END