OpenSDS: Subdivision Surfaces Library

W. Taylor Holliday

March 20, 2006

Overview

The Subdivision Surfaces Library (SDS) is a set of routines for efficient adaptive rendering of subdivision surfaces.

The SDS interface is patterned after the OpenGL Utility Library (GLU [1]). By using a simple lowest-common-denominator C interface, SDS remains language agnostic, like OpenGL. Because of the geometric complexity of subdivision surfaces, SDS operates in retained mode, unlike OpenGL, storing the specified geometry.

[Should SDS require an OpenGL context like GLU?]

A Small Example

```
/* points of a tetrahedron */
float p[4][3] = {
  {0.00000e+0, 2.00000, 0.00000e+0 },
  {0.00000e+0, -0.666660, 1.88562 },
  {-1.63299, -0.666666, -0.942810},
  {1.63299, -0.666666, -0.942810}
};

/* create a new surface */
SDS* sds = sdsNew();

/* control mesh vertices */
SDSVertexRef v[4];

/* specify the vertex locations */
v[0] = sdsVertex3fv(sds, p[0]);
v[1] = sdsVertex3fv(sds, p[1]);
v[2] = sdsVertex3fv(sds, p[2]);
v[3] = sdsVertex3fv(sds, p[3]);

/* specify the control mesh faces */
sdsTri(sds, v[0], v[1], v[2]);
sdsTri(sds, v[1], v[0], v[3]);
```

```
sdsTri(sds, v[0], v[2], v[3]);
sdsTri(sds, v[1], v[2], v[3]);

/* finally, render the surface using OpenGL */
sdsRender(sds);
```

1 The SDS Object

The SDS Object holds the state of the subdivision surface. This includes:

- tessellation parameters
- pointers to callbacks
- the control mesh and its associated attributes

To create a SDS object, use:

`SDS*` **sdsNew**();

When you are finished using the object, call:

`void` **sdsDelete**(`SDS` **obj*);

2 Callbacks

Callbacks are used to receive tessellation data from SDS or to be notified of an SDS error. To define a callback, use:

`void` **sdsCallback**(`SDS` **obj* , `SDSEnum` *which*, `void` **fn*);

Callbacks are called like one would call glBegin, glVertex, glNormal, etc. The following table summarizes the various callbacks, their prototypes, and when they are called.

which	*prototype*	*called to*
SDS_BEGIN	`void` **begin**(`GLenum` *type*);	begin a primitive
SDS_VERTEX	`void` **vertex**(`GLfloat` **vertex*);	receive a vertex
SDS_NORMAL	`void` **normal**(`GLfloat` **normal*);	set normal
SDS_COLOR	`void` **color**(`GLfloat` **color*);	set color
SDS_TEXCOORD	`void` **texCoord**(`GLfloat` **tex_coord*);	set texture coordinate
SDS_END	`void` **end**(`void`);	end a primitive
SDS_ERROR	`void` **error**(`GLenum` *errno*);	report an error

Also, SDS provides a set of callbacks which take user-specified data:

which	*prototype*
SDS_BEGIN_DATA	void **beginData**(GLenum *type*, void **userData*);
SDS_VERTEX_DATA	void **vertexData**(GLfloat **vertex*, void **userData*);
SDS_NORMAL_DATA	void **normalData**(GLfloat **normal*, void **userData*);
SDS_COLOR_DATA	void **colorData**(GLfloat **color*, void **userData*);
SDS_TEXCOORD_DATA	void **texCoordData**(GLfloat **tex_coord*, void **userData*);
SDS_END_DATA	void **endData**(void, void **userData*);

To specify *userData*, use:

void **sdsCallbackData**(SDS* *obj*, void* *userData*);

[This interface is not type-safe. Should we have an **sdsCallback** routine for each type of callback?]

void **sdsBeginFunc**(SDS **obj* , void (**f*)(GLenum *type*));
void **sdsVertexFunc**(SDS **obj* , void (**f*)(float *vertex*[3]));
void **sdsNormalFunc**(SDS **obj* , void (**f*)(float *normal*[3]));
void **sdsColorFunc**(SDS **obj* , void (**f*)(float *color*[4]));
void **sdsTexCoordFunc**(SDS **obj* , void (**f*)(float *tex_coord*[2]));
void **sdsEndFunc**(SDS **obj* , void (**f*)(void));

The following code will set up callbacks to render the surface:

```
sdsBeginFunc(sds, glBegin);
sdsVertexFunc(sds, glVertex3fv);
sdsEndFunc(sds, glEnd);
```

3 Properties

To set a propety, use:

`void` **sdsProperty**(`SDS` *obj , `GLenum` $which$, `GLdouble` $value$);

which	*value*	*default*
`SDS_SCHEME`	`SDS_CATMULL_CLARK`*[1] or `SDS_LOOP`	`SDS_CATMULL_CLARK`
`SDS_MODE`	`SDS_RENDER`* or `SDS_TESSELLATE`	
`SDS_CULLING`	`GL_TRUE` or `GL_FALSE`	
`SDS_SAMPLING_METHOD`	`SDS_PATH_LENGTH` or `SDS_PARAMETRIC_ERROR` or `SDS_DOMAIN_DISTANCE`	
`SDS_SAMPLING_TOLERANCE`	$(0, \infty)$	
`SDS_DISPLAY_MODE`	`SDS_OUTLINE` or `SDS_FILL` or `SDS_CONTROL`	

In *render* mode, primitives will be sent to OpenGL for rendering. In *tesselation* mode, the user defined callbacks will be called for each rendering primitvie.

`void` **sdsVertexProperty**(`SDS` *obj , `SDSVertexRef` $vertex$, `GLenum` $which$, `double` $value$);

which	*value*
`SDS_DART`	`SDS_TRUE` or `SDS_FALSE`

`void` **sdsEdgeProperty**(`SDS`* obj, `SDSEdgeRef` $edge$, `GLenum` $which$, `GLdouble` $value$);

which	*value*	*default*
`SDS_CREASE`	`SDS_TRUE` or `SDS_FALSE`	`SDS_FALSE`
`SDS_SHARPNESS`	$[0, \infty]$	

`SDS_CREASE` indicates a *crease-edge* with sharpness specified by `SDS_SHARPNESS`.

`void` **sdsFaceProperty**(`SDS`* obj, `SDSFaceRef` $face$, `GLenum` $which$, `GLdouble` $value$);

which	*value*
`SDS_COLOR_INTERPOLATION`	`SDS_LINEAR` or ?
`SDS_TEXCOORD_INTERPOLATION`	`SDS_LINEAR` or ?

`SDS_COLOR_INTERPOLATION` and `SDS_TEXCOORD_INTERPOLATION` specify how attributes (color, texcoords) are interpolated across the face. [does it make sense to apply the same subdivision scheme to attributes? usually attributes such as texcoords are interpolated]

To query properties, use:

`double` **sdsGetProperty**(`SDS` *obj , `SDSenum` $which$);
`double` **sdsGetVertexProperty**(`SDS` *obj , `SDSVertexRef` $vertex$, `SDSenum` $which$);
`double` **sdsGetEdgeProperty**(`SDS` *obj , `SDSEdgeRef` $edge$, `SDSenum` $which$);
`double` **sdsGetFaceProperty**(`SDS` *obj , `SDSFaceRef` $face$, `SDSenum` $which$);

4 The Control Mesh

Geometry and associated attributes are specified by the calls in this section.

4.1 Mesh Types

`typedef` *implementation-defined* `SDSVertexRef`;
`typedef` *implementation-defined* `SDSFaceRef`;
`typedef` *implementation-defined* `SDSEdgeRef`;

4.2 Specifying Geometry

SDS maintains vertex, color, and texcoord arrays which are automatically resized. Array indices start at zero. To specify or update a vertex, use:

`SDSVertexRef` **sdsVertex{34}{fd}**(`SDS` **obj* , `T` *x*, `T` *y*, `T` *z*);
`SDSVertexRef` **sdsVertex{34}{fd}v**(`SDS` **obj* , `T` **point*);

Vertices may also be specified in an array:

`SDSVertexRef` **sdsVertexArray**(`size_t` *size*, `SDSenum` type, `size_t` *stride*, `const GLvoid` *pointer)

The i^{th} vertex in the array is accessed as follows:

```
SDSVertexRef v = sdsVertexArray( ... );
SDSVertexRef vi = v + i;
```

Similar calls are provided for colors and texcoords:

`void` **sdsColor**3fv(`SDS` **obj* , `SDSVertexRef` *vertex* , `float` *color*[3]);

`void` **sdsTexCoord**2fv(`SDS` **obj* , `SDSVertexRef` *vertex* , `float` *texCoord*[2]);

4.3 Specifying Connectivity

To create a face of the control mesh, use:

`SDSFaceRef` **sdsFace**(`int` *nVertices*, `SDSVertexRef` **indices*);
`void` **sdsFaceColors**(`SDS` **obj*, `GLint` *index*, `GLint` **indices*);
`void` **sdsFaceTexCoords**(`SDS` **obj*, `GLint` *index*, `GLint` **indices*);

sdsFace returns the index of the newly created face.
Convenience functions are provided for creating triangles and quadrilaterals:

`SDSFaceRef` **sdsTri**(`SDS` **obj* , `SDSVertexRef` *a*, `SDSVertexRef` *b*, `SDSVertexRef` *c*);
`SDSFaceRef` **sdsQuad**(`SDS` **obj* , `SDSVertexRef` *a*, `SDSVertexRef` *b*, `SDSVertexRef` *c*, `SDSVertexRef` *d*);

Edges are implicit.

4.4 Mesh User-Data

`void` **sdsSetVertexData**(`SDS` *obj , `SDSVertexRef` $vertex$, `void` *$userData$);
`void` **sdsSetEdgeData**(`SDS` *obj , `SDSEdgeRef` $edge$, `void` *$userData$);
`void` **sdsSetFaceData**(`SDS` *obj , `SDSFaceRef` $face$, `void` *$userData$);

4.5 Control Mesh Iteration

`void` **sdsForEachVertex**(`SDS` *obj , `void` (*f)(`SDS` *obj , `SDSVertexRef` $vertex$));
`void` **sdsForEachEdge**(`SDS` *obj , `void` (*f)(`SDS` *obj , `SDSEdgeRef` $edge$));
`void` **sdsForEachFace**(`SDS` *obj , `void` (*f)(`SDS` *obj , `SDSFaceRef` $face$));

`SDSEdgeRef` **sdsCCF**(`SDSEdgeRef` $edge$);
`SDSEdgeRef` **sdsCCV**(`SDSEdgeRef` $edge$);
`SDSEdgeRef` **sdsEC**(`SDSEdgeRef` $edge$);
`SDSVertexRef` **sdsOrig**(`SDSEdgeRef` $edge$);
`SDSVertexRef` **sdsDest**(`SDSEdgeRef` $edge$);
`SDSFaceRef` **sdsLeft**(`SDSEdgeRef` $edge$);
`SDSFaceRef` **sdsRight**(`SDSEdgeRef` $edge$);

5 Tessellation and Rendering

To render the surface, use:

`void` **sdsRender**(`SDS` **obj*);

To apply a limit mask to a particular vertex, use:

`void` **sdsLimit**(`SDS` **obj* , `SDSVertexRef` *vertex* , `float` *point*[3]);

sdsRayCast will intersect a ray with the surface, calling the *rayHit* callback for each intersection.

`void` **sdsRayCast**(`SDS` **obj* , `float` *origin*[3], `float` *direction*[3]);

6 Errors

To get a string describing an error, use:

`const char*` **sdsErrorString**(`SDSenum` *errno*)

7 C++ Interface

[bother with this?]

```
class SubdivisionSurface {
  typedef int VertexHandle;
  typedef int FaceHandle;

  // [should templates be used to avoid dynamic dispatch?]
  class BeginFunctor {
    virtual void operator()(GLenum type) = 0;
   };

  class VertexFunctor {
    virtual void operator()(GLdouble* v) = 0;
  };

  // etc...

  void beginCallback(BeginFunctor& f);
  void vertexCallback(VertexFunctor& f);

  // etc...

  void property(GLenum which, GLdouble value);
```

```
  void render();

  static const char* errorString(GLenum errno);
};
```

References

[1] *The OpenGL Utility Library*

[2] Jos Stam. *Exact Evaluation Of Catmull-Clark Subdivision Surfaces At Arbitrary Parameter Values*

[3] Tony Derose, et al. *Subdivision Surfaces in Character Animation*

[4] Jeffrey Bolz and Peter Schroder. *Rapid Evaluation of Catmull-Clark Subdivision Surfaces*