Rendering Subdivision Surfaces on Graphics Hardware

Avi Bleiweiss
Silicon Graphics Inc.
September 4th 1996

Motivation

Subdivision is a powerful paradigm for the generation of arbitrary topology. Direct representation of a surface provides an attractive option for fast rendering.

The detailed simulation of related algorithms will lead to an efficient pipe design that supports subdivision well.

Agenda

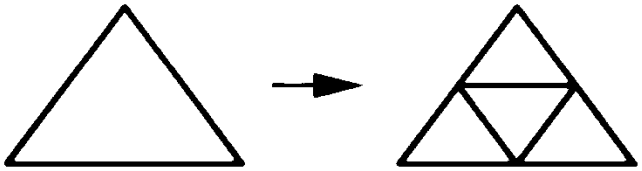
- Background
- OpenGL API
- Implementation
- Performance
- Realization

Background

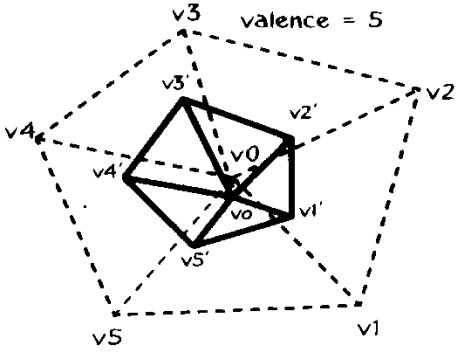
- Repeated refinement of an initial control mesh
- Doo/Sabin and Cutmull/Clark [78] used quadrilateral meshes
- Loop [87] introduced a simple triangulated mesh scheme that leads to a tangent plane smooth surface (C1)
- Interpolating schemes by Dyn et al. [90] and Zorin et al. [92]
- Multi-resolution analysis

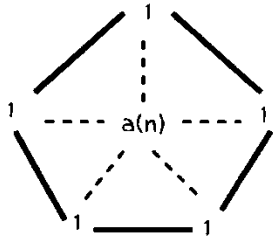
Background (cnt'd)

- Loop's refinement step - splitting each triangle by four

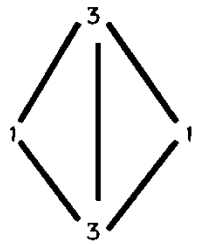


- Vertices of a refined mesh are computed as an affine combination of vertices in the unrefined one





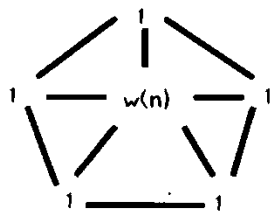
vertex mask



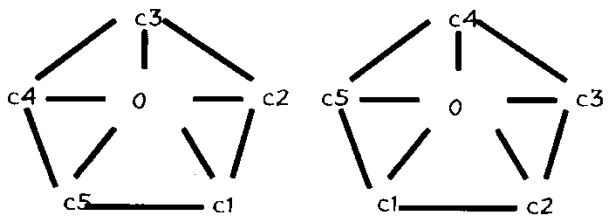
edge mask

Background (cnt'd)

- Subdivision surfaces are defined as the limit of an infinite refinement process
- Limit point can be expressed as an affine combination of initial vertex position
- Tangent vectors to the limit surface can be easily extracted and normals are computed using cross-product of tangents



position mask



tangent mask

Background (cnt'd)

- Hoppe et al. [94] expanded subdivision rules to accurately model an object with tangent discontinuities
- Edges can be either smooth or sharp
- Vertices are classified as either smooth, dart, crease, or corner whether they have zero, one, two or more sharp incident edges, respectively
- Crease vertex is regular if there are exactly two smooth edges between the sharp ones. Otherwise - irregular
- All subdivision rules are expressed as masks, are flexible and extensible
- Subdivision rules apply to both positions and attributes

Background (cnt'd)

- Key applications:

  - Arbitrary mesh paradigm is a key strength and direct representation avoids trimming bottleneck in rendering

  - Geometry compression is evident by the virtue of refinement. A typical level of 3 or 4 can yield ratios in the range of 8 to 32

  - Continuous LOD control is accomplished by interpolating adjacent subdivision levels (avoids storing geometry at different LOD's at the application space)

Background (cnt'd)

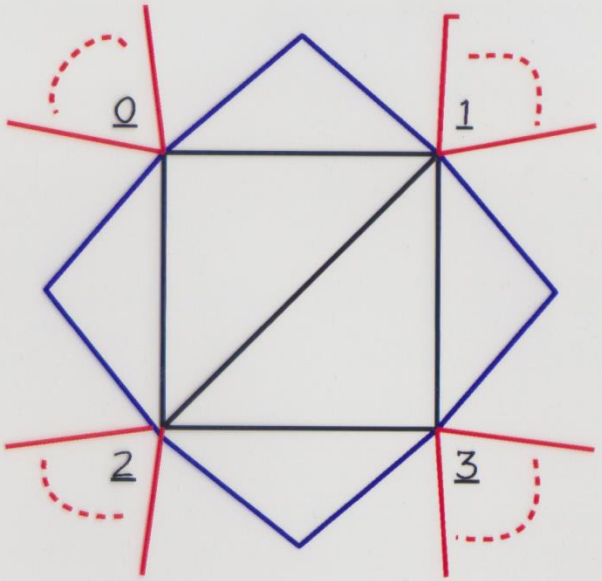
- Kari Pulli and Mark Segal work: mesh editor, and sliding window subdivision scheme - basis for the effort presented

- Goals:

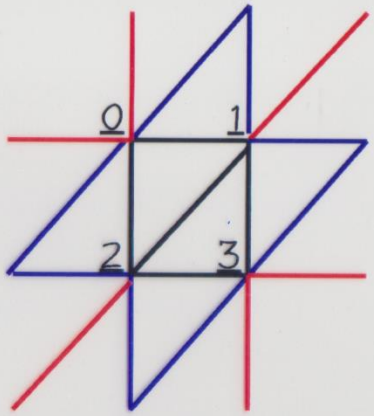
  - OpenGL API proposal, transport layer definition for IR
  - Complete software implementation of sliding window subdivision method
  - Add LOD morph capability
  - Evaluate portability and performance in a GE (like) domain

OpenGL API

- New primitive into the pipe: tripair patch



generic tripair patch



tstrip tripair patch

- The tripair patch topology is composed of a tripair core (blk), neighboring triangles (blu) and corner support edges (red)

OpenGL API (cnt'd)

- Considerations:
  - Discrete vs pointer based commands
  - Minimal impact on user, provision for caching of patches
  - Patch definition to allow for topological degeneracy
  - Distinguish smooth from non-smooth patch for chosing a fast computation path
- Simpler to go with pointer based commands
- Trifan per-corner definition for vertex data, vertex and edge tags (ccw)
- SGIX_subdiv_patch extension

OpenGL API (cnt'd)

- Patch control Commands:

  glSubdivPatchParameterfSGIX(GLenum pname, GLfloat param);
  glSubdivPatchParameteriSGIX(GLenum pname, GLint param);

  These commands set refinement level (float for LOD morph) and mesh format.

  - pname takes:

    GL_SUBDIV_PATCH_LEVEL_SGIX
    GL_SUBDIV_PATCH_FORMAT_SGIX

  - param for format takes:

    GL_V3F
    GL_C4UB_V3F
    GL_T2F_V3F
    GL_T2F_C4UB_V3F

- The user can query refinement level, mesh format, and maximum valence allowed for the implementation

OpenGL API (cnt'd)

- Patch data command:

```
glSubdivPatchSGIX(const GLvoid **vertexData,
                  const GLvoid **vertexTags,
                  const GLvoid **edgeTags);
```

Tag data takes:

GL_SUBDIV_PATCH_SMOOTH_SGIX
GL_SUBDIV_PATCH_CREASE_REGULAR_BIT_SGIX
GL_SUBDIV_PATCH_CREASE_IRREGULAR_BIT_SGIX
GL_SUBDIV_PATCH_DART_BIT_SGIX
GL_SUBDIV_PATCH_CORNER_BIT_SGIX
GL_SUBDIV_PATCH_CONICAL_BIT_SGIX
GL_SUBDIV_PATCH_SHARP_BIT_SGIX

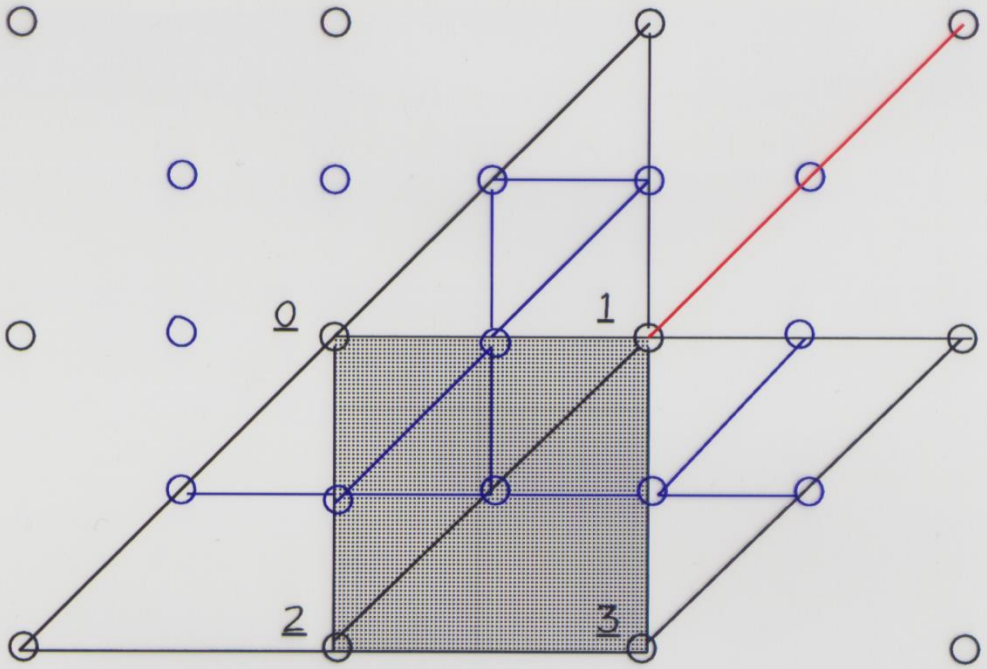
- Smooth patch is denoted by null's for both the vertexTags and edgeTags pointers

Implementation

- Patch topology is on a 2D grid
- Naive approach to subdivision uses memory in the order of output size
- Sliding window subdivision scheme facilitates incremental subdivision, balanced with tstrip output rendering
- Three rows worth of strided vertex data memory is required for each level (scratch) plus two rows for tstrip output (fifo)
- Maximum refinement level supported is 4
- Sliding window method is partitioned into initialization and core phases

Implementation (cnt'd)

- At initialization, three rows of vertex data are generated per refinement level



Initialization (l0 (blk) -> l1 (blu))

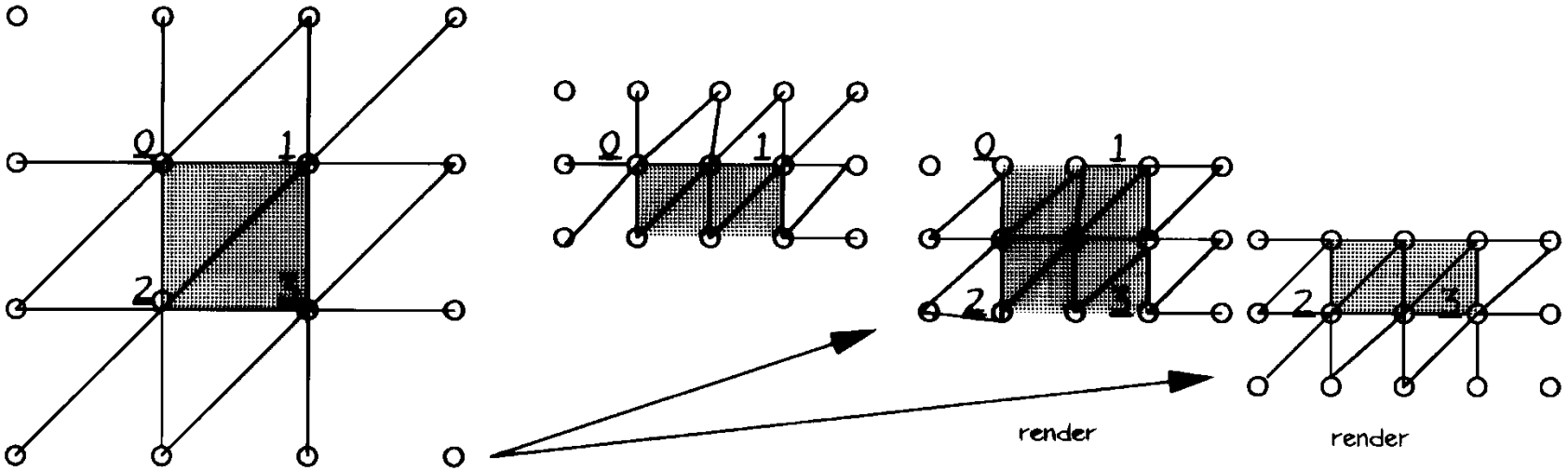
Implementation (cnt'd)

- At the conclusion of Initialization, one row of vertices is ready for output
- Incremental row is generated by either subdividing vertical/diagonal edges or by subdividing horizontal edges and updating old vertices
- A newly generated row at level j produces two new rows in level j+1
- The core code resembles unrolled recursion (maximum refinement level is 4)
- Pseudo code for sliding window core (last two levels):

```
while(row[level-1] < t[level -1] {
    DoRow(level-1)
    t[level] = row[level] + 2;
    while(row[level] < t[level] {
        DoRow(level);
        DoRender(level);
    }
}
```

Implementation (cnt'd)

- Illustration of sliding window method core:



level 0

level 1

Implementation (cnt'd)

- For each incremental row that hits the finest refinement level, as set by the user:

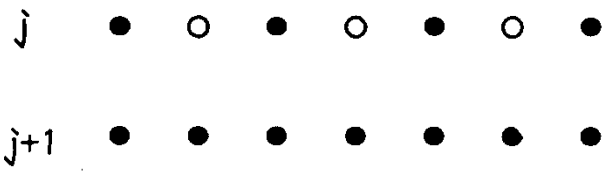
  - Limit surface positions are computed

  - Once GL_AUTO_NORMAL is enabled, tangents and normals are computed

  - Morph LOD:

    - Finalize incremental row for one before finest refinement level

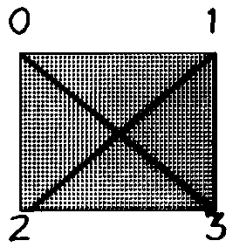
    - Linearly (or bilinearly) interpolate between two rows of adjacent refinement levels, using an LOD transfer function

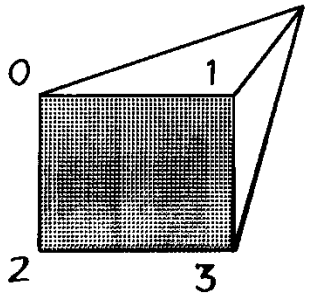


Implementation (cnt'd)

- Topological degeneracy is transparent to the subdivision process. It only affects OpenGL's transport layer or the final rendering part

- Examples:





- No mate tripair is refined as a normal one and is bounded by its diagonal at the output

- Gracefully handle exceeding maximum valence

Implementation (cnt'd)

- Memory and arithmetic characteristic of sliding window method:

  - Scratch area (words) to keep three rows for each refinement level as a function of mesh format (assuming four refinement levels):

GL_V3F	414+30n
GL_C4UB_V3F	966+70n
GL_T2F_V3F	640+50n
GL_T2F_C4UB_V3F	1242+90n

n - maximum support corner vertices (6 is minimum and 8 is decent)

  - A fifo worth of 34 vertices is needed to keep two output rendering rows (morph LOD requires 51 vertices)

  - Mask math is coefficient table driven, 256 table entry is adequate

  - Unique instructions: integer modulus (for addressing, 5 bits suffices), inverse-sqrt (for computing normalized normals)

Performance

- Benchmark ran on a 196MHz R10K (ver 2.4) with 1MB cache, three levels of refinement (128 tris per patch)

- Figures in table reflect subdivision compute speed (In parents, IR rendering numbers per-GE):

	Kpatches/sec	MTstrips/sec
no light, no texture	5.2	.667 (2.0)
light, no texture	4.5	.576 (1.5)
no light, texture	3.7	.480 (1.5)
light, texture	3.2	.419 (1.0)

- Subdivision to geometry pipeline performance ratio is 1 : 3

Realization

- Driver software:

  - Subdivision surfaces - new surface representation

  - View dependent LOD for geometry: single representation of geometry per-frame, reduction of detail, figure out LOD transfer function (Performer, Jenny)

  - Tessellation assist: cuts down computation on the host, reduces bandwidth to the pipe, conversion of representations (Inspector, Brian C.)

  - NURBS: support for both subdivision and evaluation inside the pipe (Zicheng)

Realization (cnt'd)

- Pipe:

  - Subdivision surfaces representation trades off bandwidth and memory with computation

  - Transport layer: atomic definition for state embedded primitive

  - Patch distribution out of a resident patch cache

  - Inter and intra patch parallelism: patch per processor or patch across many processors, respectively

  - Load balance: front vs back end of the pipe

Realization (cnt'd)

- IR:

  - Patch definition split into two pipe commands: one for vertex data and one for state (tags and flags)

  - Scratch size inside the GE limits finest refinement level to three for some of the formats. Fifo size limits maximum corner support vertices

  - Sliding window scheme can explore SIMD to a certain extent (dominant patch boundary computations are single cored)

  - Save/restore scratch data per incremental row

  - Mask coefficients and integer modulus tables stored in Eram

Realization (cnt'd)

- Future:

  - Atomic, variable length patch definition into the pipe

  - Dedicated scratch size of 2K words and fifo worth of 50 vertices per-patch, close to a floating point core.

  - Parallel cores fed from a staging area will edge SIMD. Can leverage off a dot product engine

  - Fast mask coefficient table access, integer modulus instruction, single cycle bit test, address registers (24)

  - Criteria for success: single node GE 3x faster than the Beast

Summary

- Conclusions

  - Loop's subdivision surfaces representation is simple, direct and extensible

  - Sliding window method for subdivision fits OpenGL's ucode paradigm, once memory constraints are met

  - Parallelism is preferred over SIMD

  - Mask driven computation are matrix based and can leverage off a dot product engine realization

  - We need more exposure to high level software (reference implementation ported to IR, RE and Indigo)