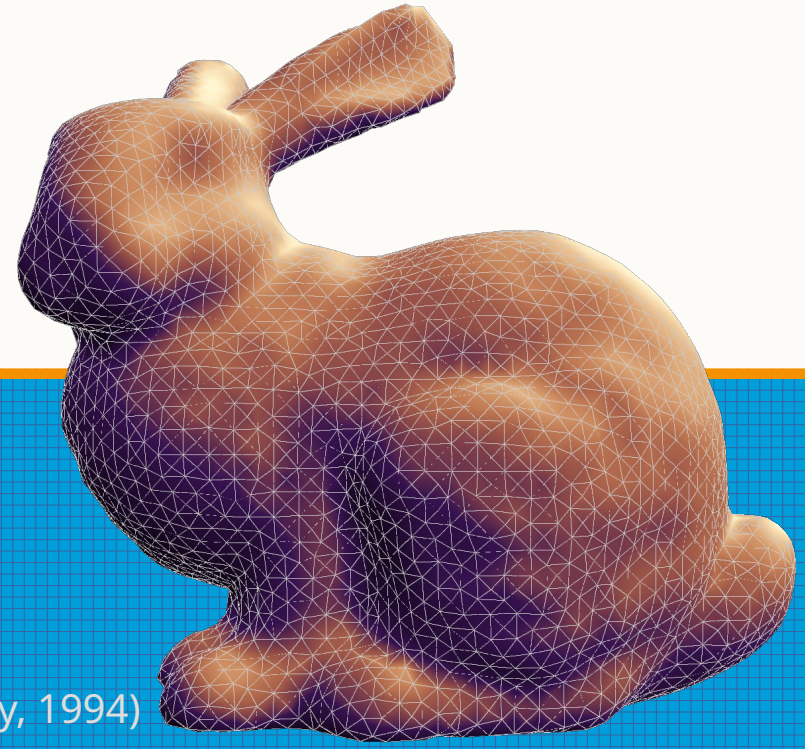
An Examination of Isosurface Extraction for 3D Terrain Generation in Videogames

by Cassette Costen

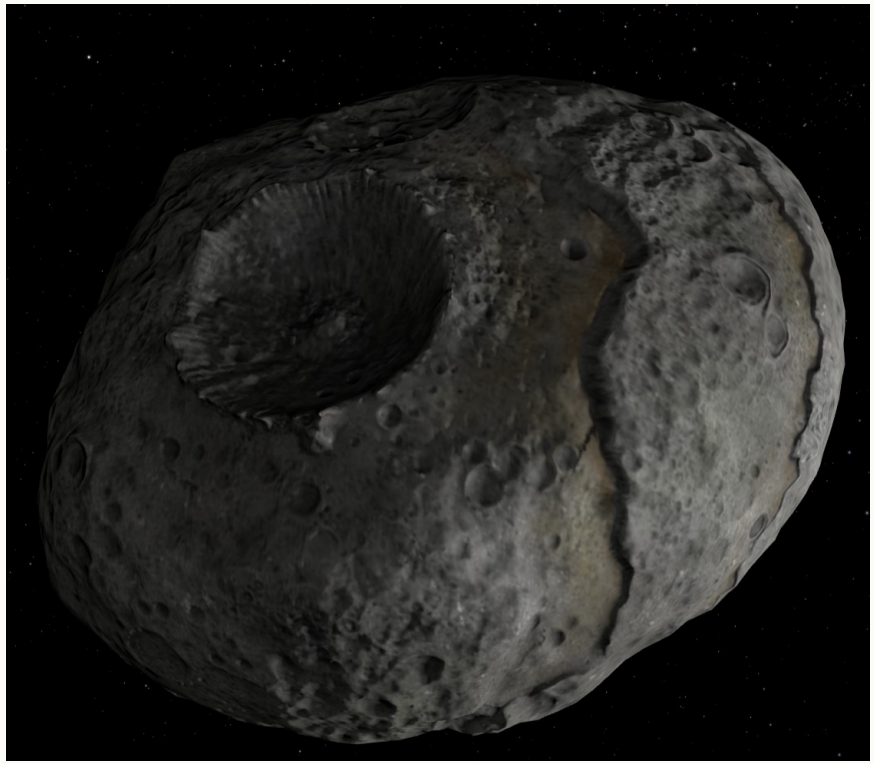


featuring the Stanford Bunny (Turk and Levoy, 1994)

Why?

To be able to generate beautiful, non-height-map 3D terrain for videogames, at runtime!

ASTRONEER (2016)



Asteroid ‘16 Psyche’ (from NASA Solar system explorer, https://science.nasa.gov/solar-system/asteroids/16-psyche/)

Asteroid ‘433 Eros’ (from Wikipedia, image origin https://photojournal.jpl.nasa.gov/catalog/PIA02923)

Actual Goals

For the algorithm?

- Mesh generation
- Primarily for procedural noise functions
- Near-realtime performance
- Minimal triangle count
- Even triangles
- Modular (chunks)

For the dissertation?

- Compare tessellation schemes
- Compare merging/ regularisation methods
- Measure triangle quality & count
- Measure performance
- Measure ‘visual beauty’ of resulting mesh

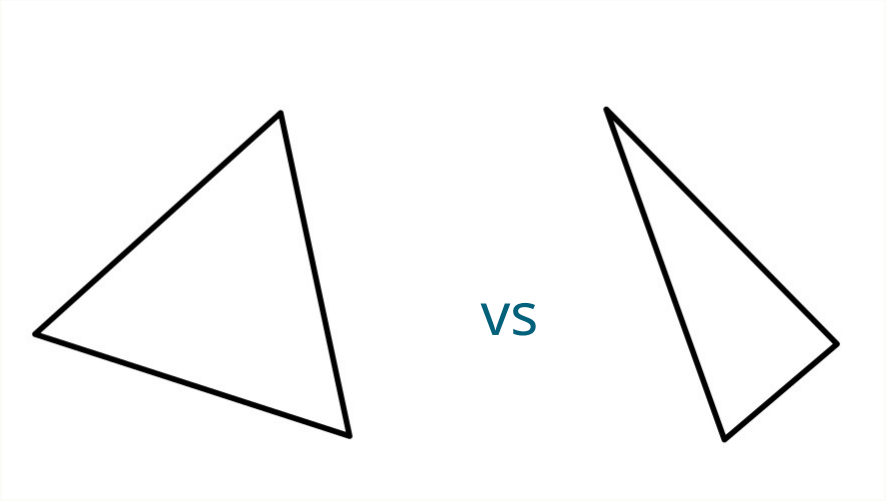
Why these goals?

We want to generate meshes which look good, and are usable in videogames without additional intervention.

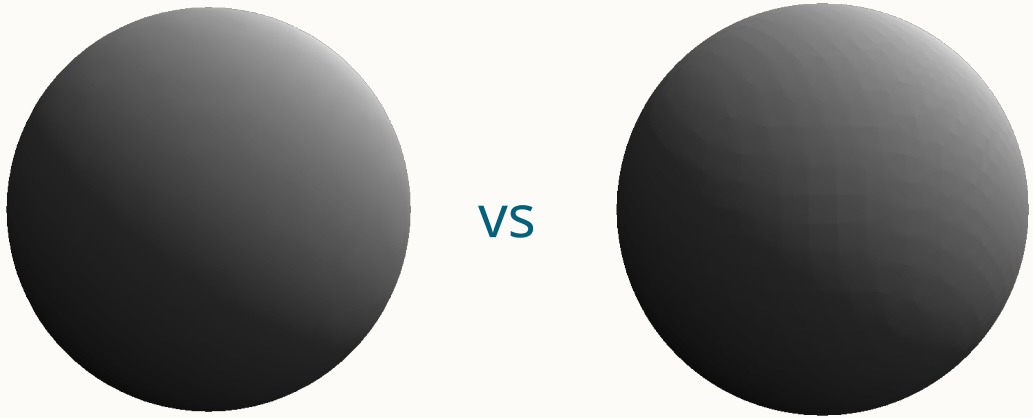
- Minimal triangle count – reduce rendering/storage cost
- Even triangles – smooth shading, or otherwise ‘pleasant’, regular appearance
- Manifold mesh – no holes in the surface, continuous surface direction
- High-performance – we may need to update the mesh very frequently, perhaps even every frame
- Modular – we may not need to update the whole terrain mesh at once, perhaps only a small part needs to be recalculated

How do we quantify this?

Triangle quality or visual beauty are abstract concepts – but we can quantify them!

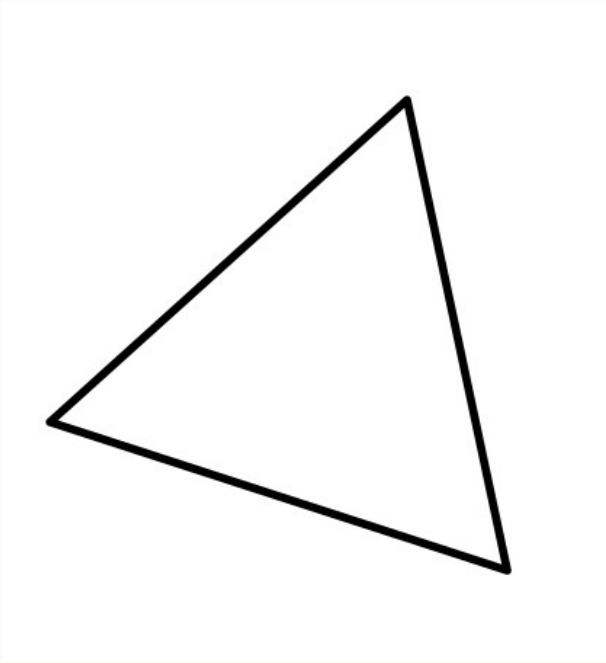


Objective – aspect ratio!



Subjective – human preference!

We want this:



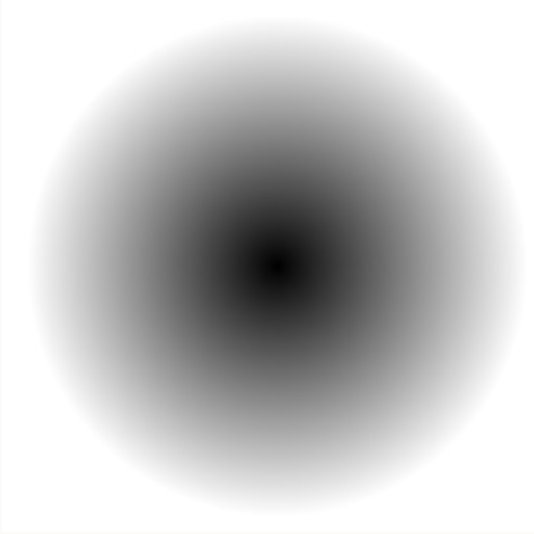
END OF INTRODUCTION

Isosurface Extraction Basics

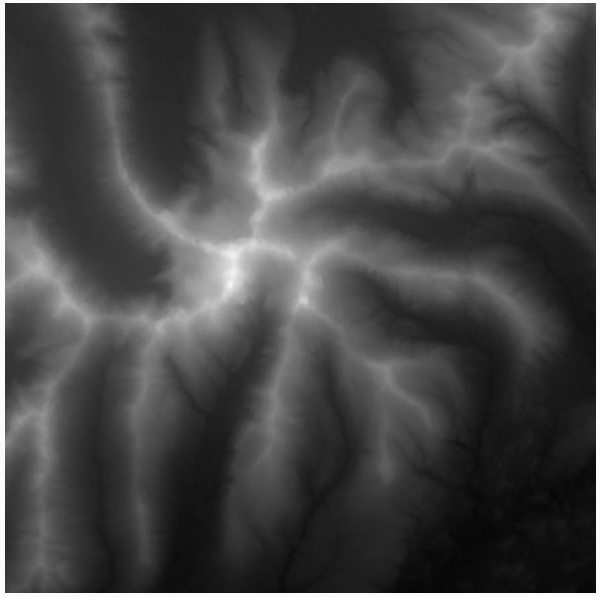
What we need:

- A continuous scalar field, or a function describing one
- A threshold value

The **isosurface** is the manifold separating the regions where the scalar field takes on a value less than the threshold, from the region where the value is greater.

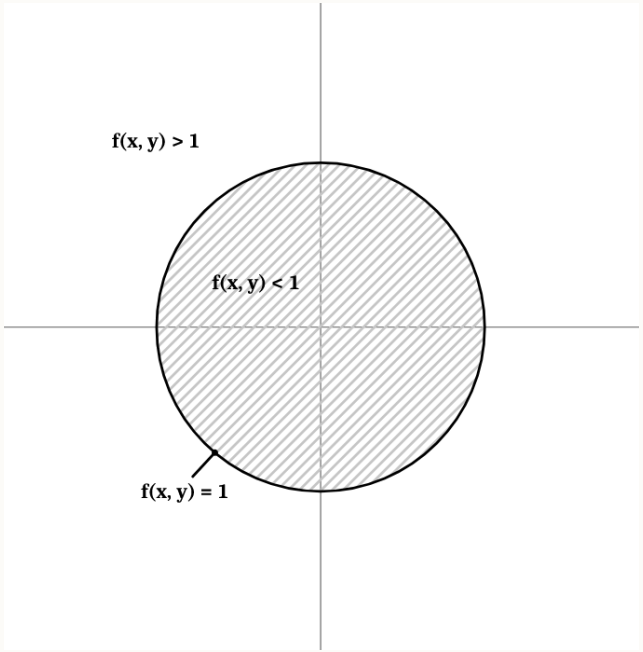


$$f(x, y) = \sqrt{x^2 + y^2}$$



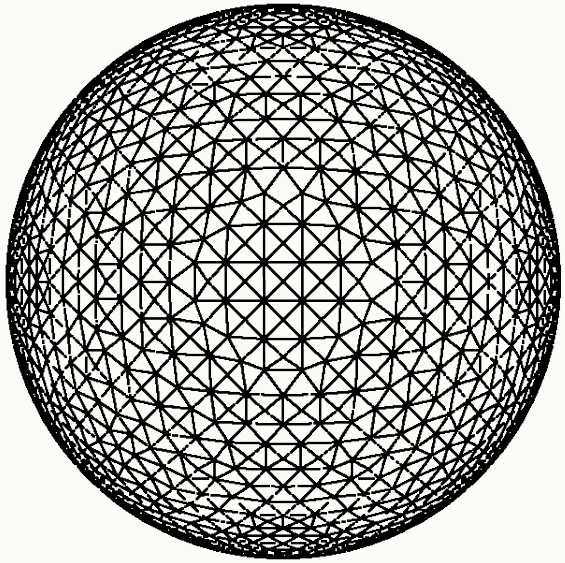
Isosurface Extraction Basics

For example…. a circle is a 1-manifold defined by $f(x, y) = \sqrt{x^2 + y^2}$ with a threshold value equal to the circle's radius (e.g. *r* = *1*)!



Isosurface Extraction Basics

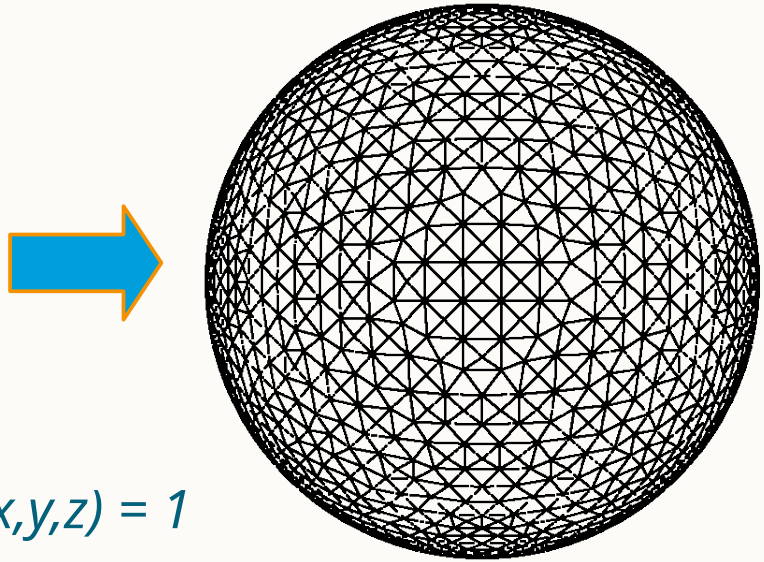
Likewise a sphere is a 2-manifold (a locally Euclidian surface with 2 degrees of freedom), embedded in 3D space, and defined by a function taking 3 positional components (*x, y, z)*.



Isosurface Extraction Basics

Isosurface extraction is the process of turning an abstract isosurface into a triangle mesh.

$$f(x, y, z) = \sqrt{x^2 + y^2 + z^2}, \text{ where } f(x, y, z) = 1$$



Every vertex has *f(x,y,z) = 1*

but how do we do that?

Key research paper:
Treece, Prager, and Gee (1999) – Regularised Marching Tetrahedra

there are some basics first though

Marching Cubes – Lorensen and Cline (1987)

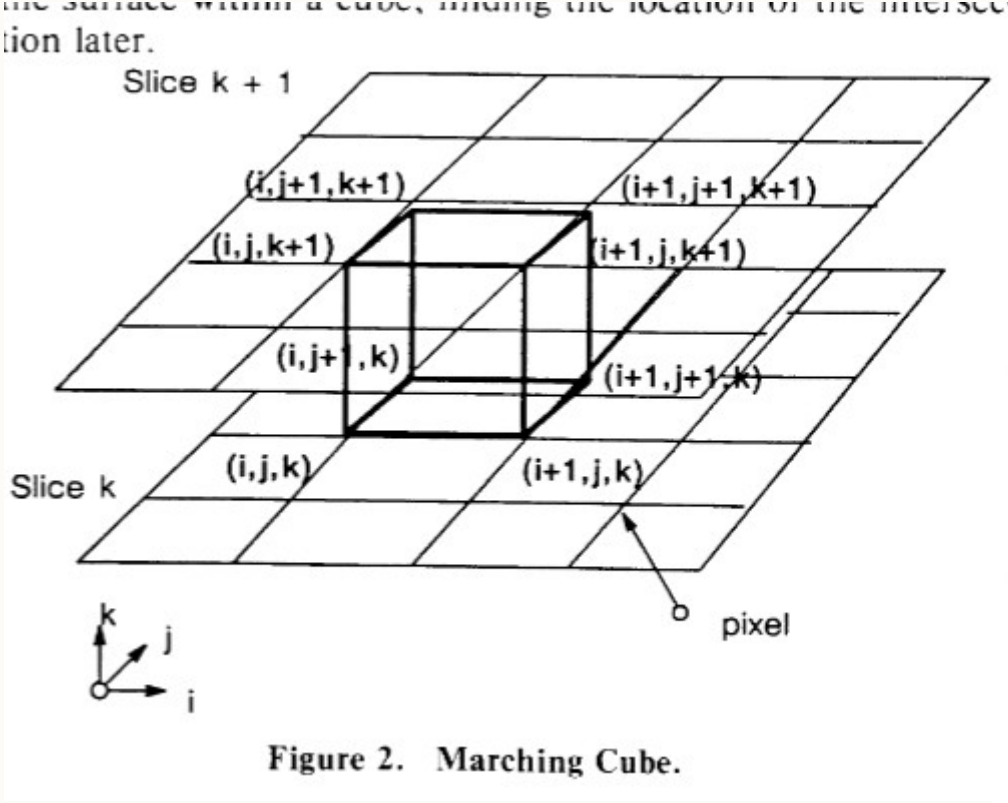


Figure 2. Marching Cube.

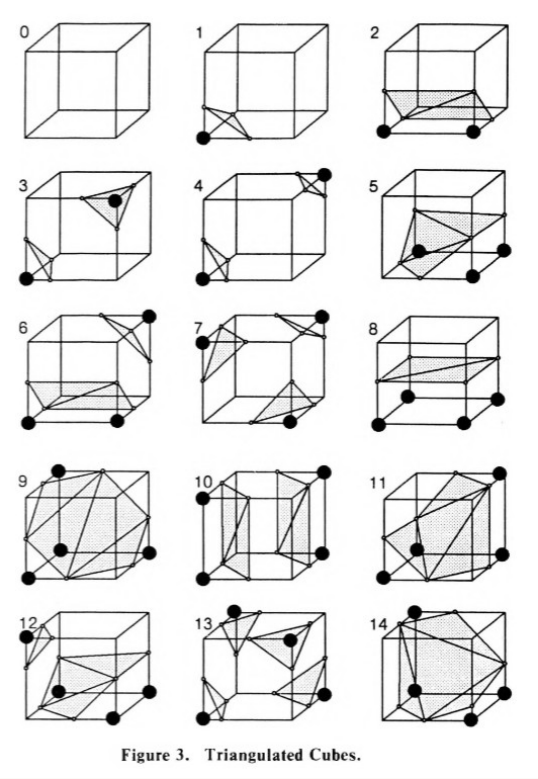


Figure 3. Triangulated Cubes.

- Divide and conquer
- Check which corners are inside or outside the isosurface
- Select triangulation accordingly
- Issues with ambiguity
- Lots of configurations
- Makes lots of tiny/elongated triangles

Diagrams sourced from Lorensen and Cline (1987)

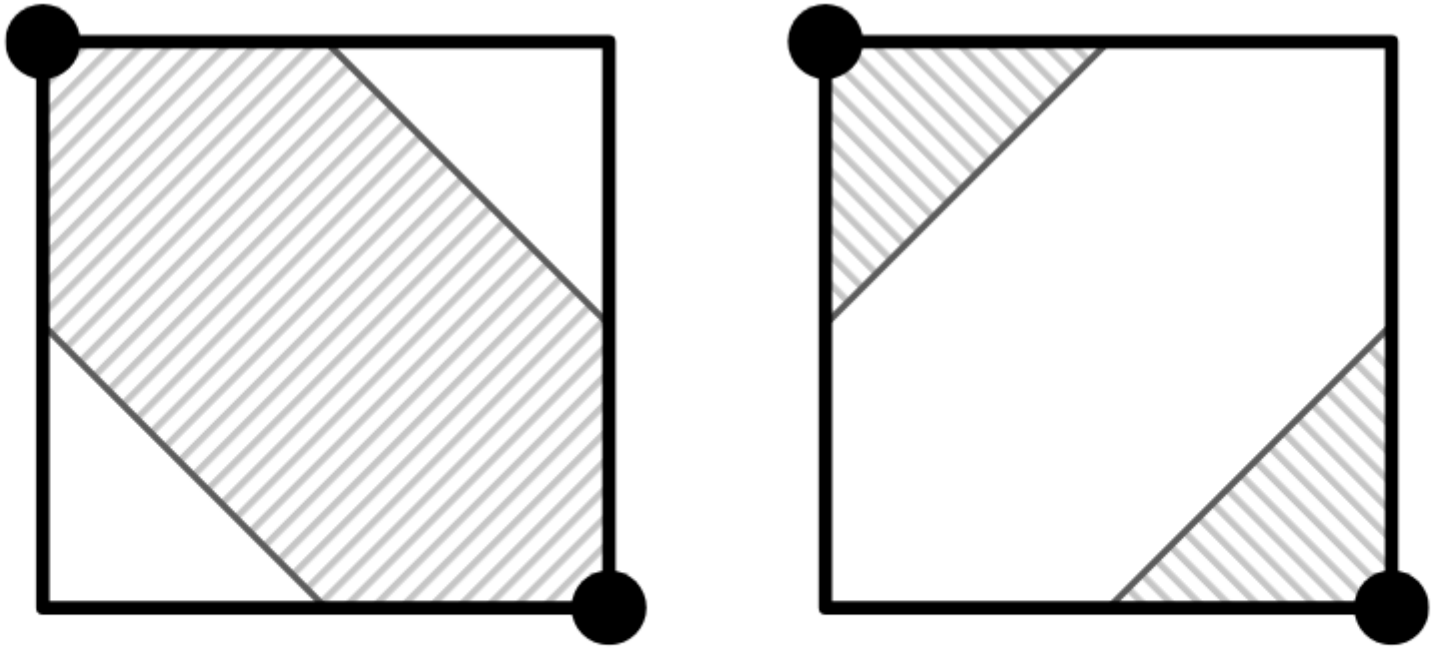


Figure 5: Illustration of the ambiguity problem with marching cubes. Without additional information, it cannot be determined which of the two triangulations is appropriate, which results in holes between cubes when extended into 3D.

Zhou et al. (1995)

Gallagher and Nagtegaal (1989)

Nielson and Hamann (1991)

Marching Tetrahedra - Payne and Toga (1990)

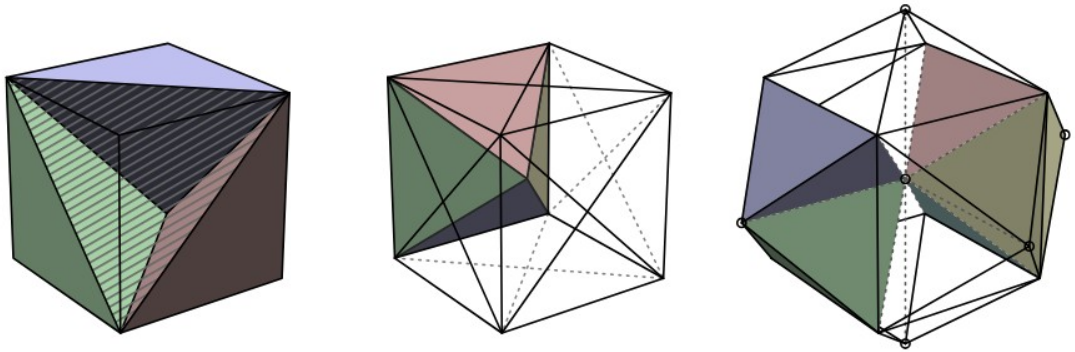
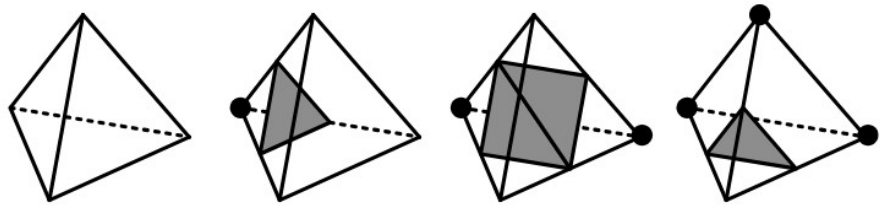


Figure 7: Left: Simple cubic arrangement, containing 5 tetrahedra in total, with 3 coloured solid.
Center: Face-centered arrangement, containing 24 tetrahedra in total, with 4 coloured solid.
Right: Body-centered arrangement, containing 24 tetrahedra in total, with 5 coloured solid. Note that each tetrahedron is a member of two adjacent cubes, meaning each cube effectively contains 12 tetrahedra. Although this arrangement is complex to depict, it is highly effective for optimising triangle shape and count.

- Tetrahedra instead of cubes (only 4 corners per primitive)
- Only 16 configurations, which reduce to only 2
- No ambiguity
- Different tetrahedral tessellations
- The body-centric arrangement has a better edge length ratio (Chan and Purisima, 1998)
- Often makes even worse triangles



how do we fix the triangles?

regularisation – aka merging!

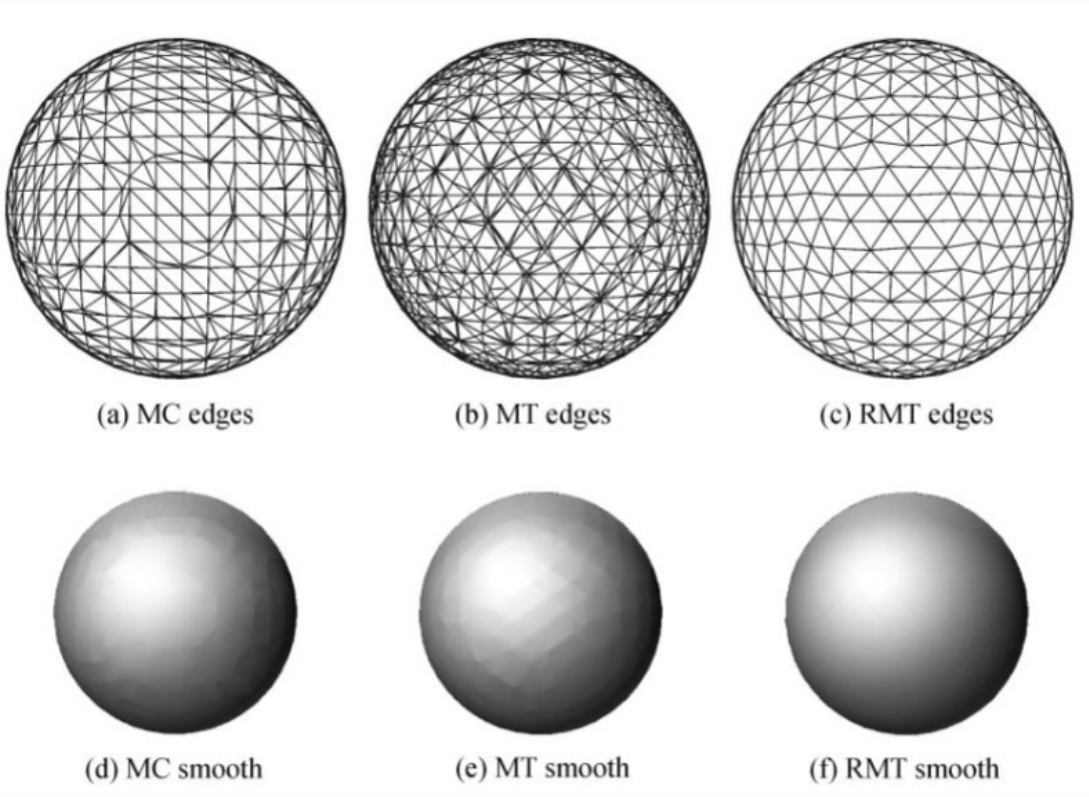
- Aspect ratio should be as close to 1 as possible
- Fewest triangles possible
- Deviation in area and aspect ratio of triangles should be minimal

can we just merge-by-distance the mesh?

1) Slow (really really really slow!)
2) Doesn’t preserve the topology of the mesh
3) Requires more memory

As explained in Treece, Prager, and Gee (1999)

Regularised Marching Tetrahedra – Treece, Prager, and Gee (1999)



- An extension of MT
- Uses the body-centered arrangement
- Regularises triangles during creation
- Preserves topology using available information
- Super fast
- Excellent smooth shading
- Highly regular triangles with great aspect ratio
- Massively reduced triangle count (70-80%)

$F_{discard}$	AR_{mean}
0.000%	775.93
79.44%	2.16

Illustration from Treece, Prager, and Gee (1999)

What about other extraction methods?

- **Dual contouring** – create one vertex per cube, position it on the isosurface, link it to vertices in neighbouring cubes (Ju et al., 2002). Requires gradient information, usually combined with octrees (Liang and Zhang, 2013).
- **Advancing front** – incrementally ‘grow’ the triangulation across the surface (Mueller, Roe, and Deconinck, 1992). Highly adaptable, can take advantage of gradient information if available. Can be highly regular.
- **Recursive subdivision** and **octrees** – instead of a fixed resolution grid like with MC, MT, RMT, the grid can be subdivided according to the density of surface detail (i.e. as described by the gradient within each cube). Allows the algorithm to save time on simple areas while preserving high frequency detail. Doesn’t generate particularly regular meshes (Bey, 1995).

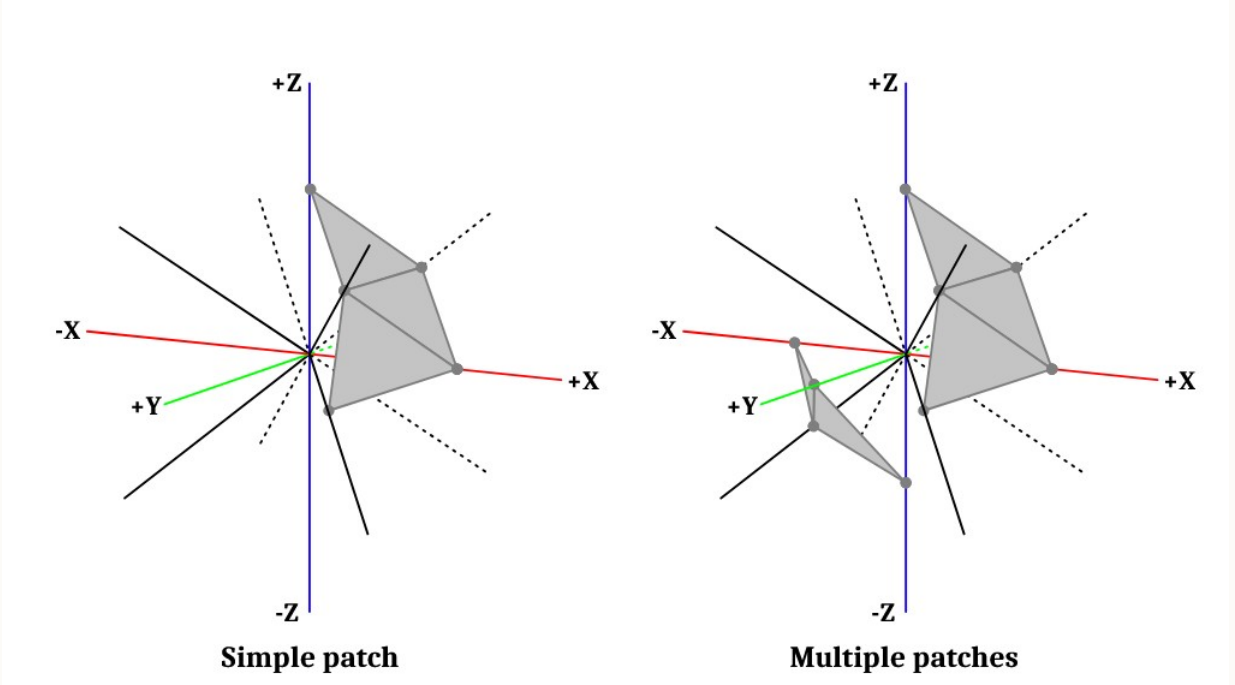
What about other extraction methods?

- **Adaptive grids** and **Delaunay triangulation** – again, instead of a rigid, fixed grid, the grid may be adjusted before isosurface extraction begins in each cell (Binninger et al., 2025). Often makes use of an ‘energy function’ to smooth out the grid to best cover the isosurface, without wasting time and memory on empty cubes. Also highly regular, but requires iterative processing.

Regularisation Techniques

We want to reduce the triangle count and increase the regularity of triangles, while preserving topology.

What does preserving topology actually mean?



Regularisation Techniques

- Surface re-tiling (Turk, 1992) – mix of isosurface extraction and simplification; creates evenly distributed new points on an existing mesh, then mutually re-tessellates the mesh in localised patches to preserve topology, until all original vertices have been eliminated.
- Could be employed on a mathematical surface, taking advantage of gradient information (at this point, essentially a Delaunay triangulation).

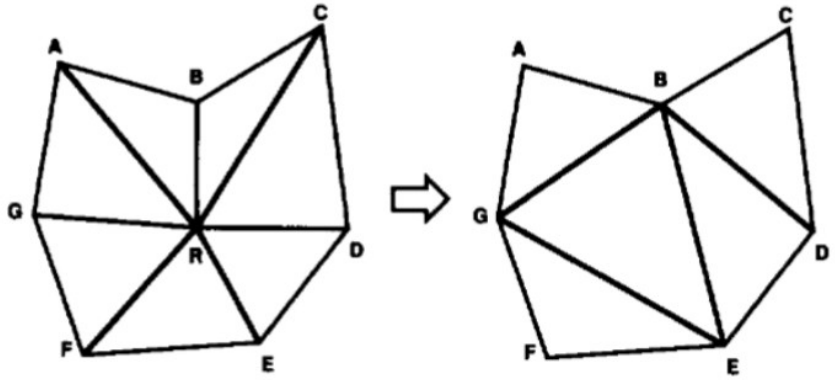


Figure 11: Example of a local re-triangulation to eliminate the vertex R (Turk, 1992).

Merging Techniques

- Merge by distance – compare every vertex against every other, and merge them into one (deleting relevant triangles) if the distance between them is less than the threshold. Terrible at preserving topology, and super super slow (Cignoni et al., 1998).
- Coplanarisation – flat areas of the mesh can be simplified into fewer triangles. Preserves topology but can be expensive, and doesn’t produce regular triangles, especially for organic meshes.
- Edge collapse – similar to merge-by-distance, except only existing edges are permitted to be merged. Mostly preserves topology (although complex features such as small loops may not survive), and requires much less processing than merge-by-distance.

END OF LITERATURE REVIEW

How RMT Actually Works (it's magic)

1. Compute value of function at each sample point

2. Check which neighbours are on the opposite side of the threshold (intersections)

3. Merge intersections based on topology, and create vertices accordingly

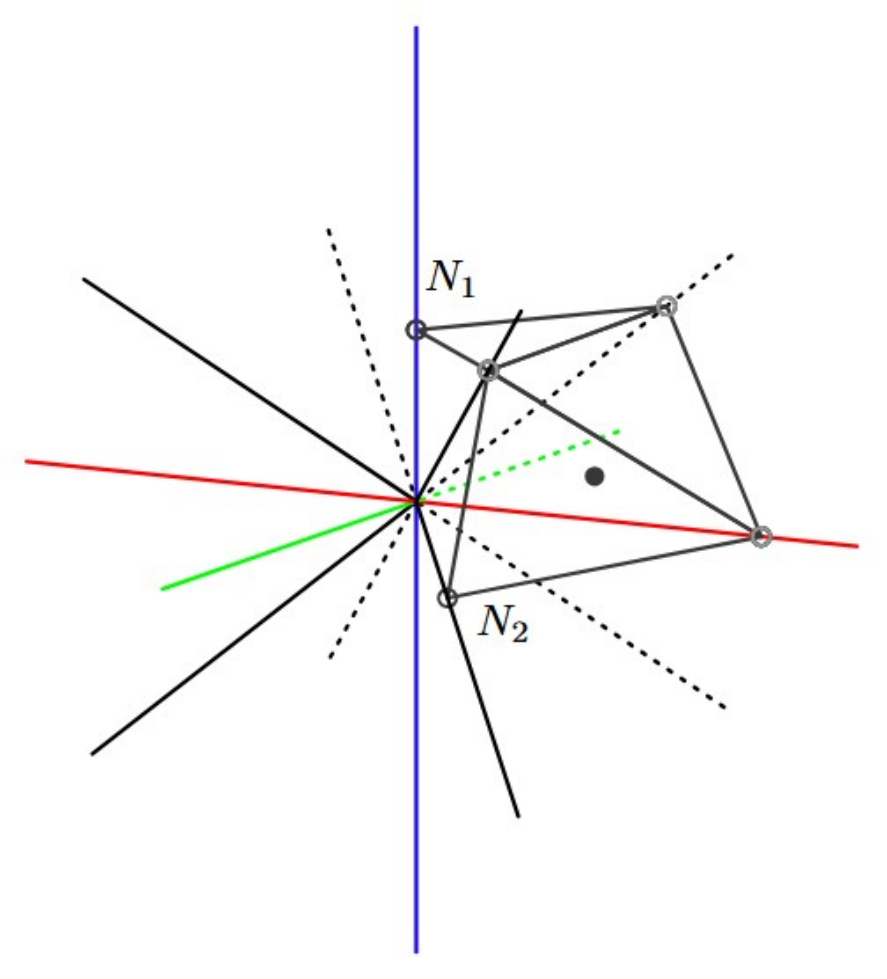
4. Use stored vertex references to construct geometry per-tetrahedron

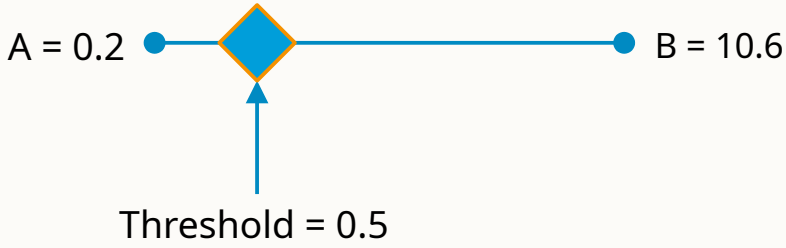
Algorithm 1: Sampling stage

Let **F** be a smoothly defined scalar function on x, y, z
for *each sample point in the lattice* **do**
    Let **P** be the position in space of the sample point
    Evaluate **F** for the given point **P**, storing the result at the sample point
end

Algorithm 2: Vertex creation & clustering stage

```
Let F be a smoothly defined scalar function on x, y, z
Let T be the threshold value for which the isosurface is being computed
Let A_v be the array of vertices in the generated mesh
for each sample point in the lattice do
    Let C equal the value of F retrieved from the current sample point
    Let S_1 ... S_14 equal the values of F retrieved from the neighbouring sample
     points in the lattice
    Let I be a series of Boolean flags representing isosurface intersections
     surrounding the current sample point
    for each neighbouring sample point do
        if sign(C − T) ≠ sign(Sn − T) and ‖C − T‖ < ‖Sn − T‖ then
            Set the flag in I corresponding to neighbour n
        end
    end
    Store the value of I at the current sample point
    Using I, construct a graph G representing which of these intersections can
     be merged together
    if G contains loops then
        Skip to the next sample point
    else
        Check for disconnected islands in G
        for each island in G do
            Create a single vertex R at the average position of all the isosurface
             intersections in the island
            Append R to the vertex array A_v and let V be its index within the
             array
            for each intersection in the island do
                Store V on the corresponding edge
            end
        end
    end
end
```





Are the two sample values on opposite sides of the threshold? (yes)

*Therefore there is an intersection, a '**candidate vertex**' on this edge.*

Which sample point is closer to the **candidate vertex**? (A)

*A gets a bitflag set marking this edge as having a '**nearby intersection**'.*

DO THIS FOR ALL 14 EDGES

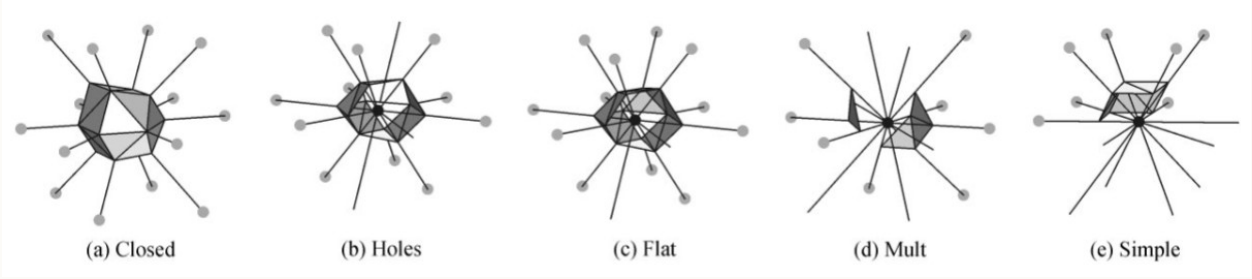


Illustration from Treece, Prager, and Gee (1999)

Now we ask – “can these intersections be treated as just one?”
The answer is “not always” – it depends on the topology!

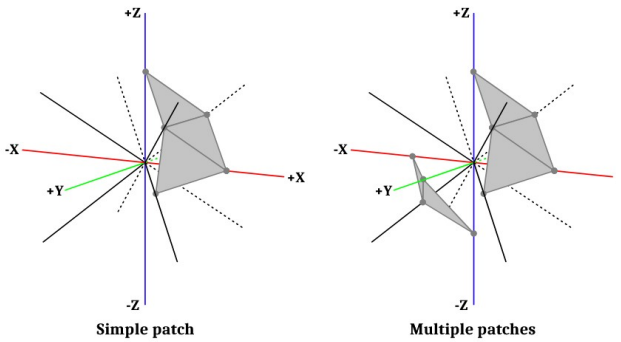


Figure 13: Two mergeable candidate topologies; note that the multiple patches exam must have each patch/island merged individually.

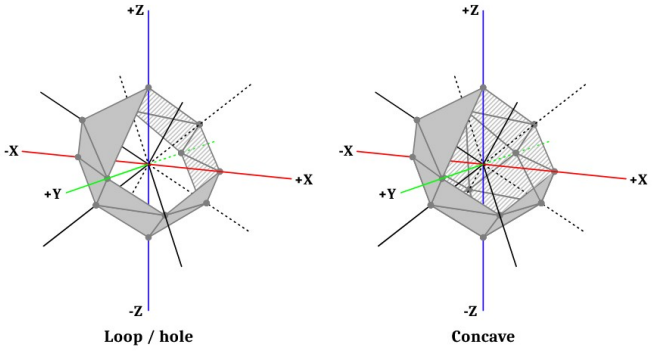
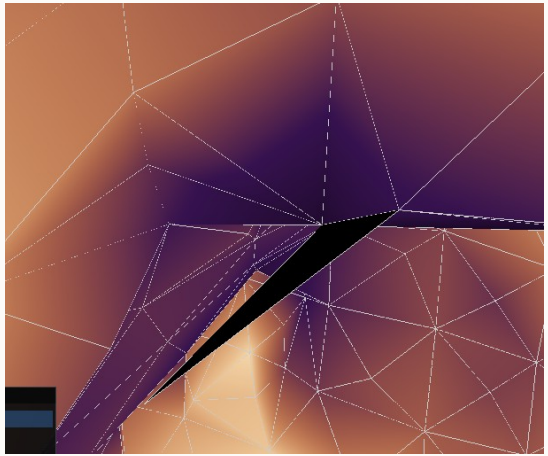


Figure 14: Two non-mergeable candidate topologies.



Merging things we shouldn’t will lead to altered topology, and often invalid/non-manifold meshes….

Algorithm 2: Vertex creation & clustering stage

```
Let F be a smoothly defined scalar function on x, y, z
Let T be the threshold value for which the isosurface is being computed
Let A_v be the array of vertices in the generated mesh
for each sample point in the lattice do
    Let C equal the value of F retrieved from the current sample point
    Let S_1 ... S_14 equal the values of F retrieved from the neighbouring sample
     points in the lattice
    Let I be a series of Boolean flags representing isosurface intersections
     surrounding the current sample point
    for each neighbouring sample point do
        if sign(C − T) ≠ sign(Sn − T) and ‖C − T‖ < ‖Sn − T‖ then
            Set the flag in I corresponding to neighbour n
        end
    end
    Store the value of I at the current sample point
    Using I, construct a graph G representing which of these intersections can
     be merged together
    if G contains loops then
        Skip to the next sample point
    else
        Check for disconnected islands in G
        for each island in G do
            Create a single vertex R at the average position of all the isosurface
             intersections in the island
            Append R to the vertex array A_v and let V be its index within the
             array
            for each intersection in the island do
                Store V on the corresponding edge
            end
        end
    end
end
```

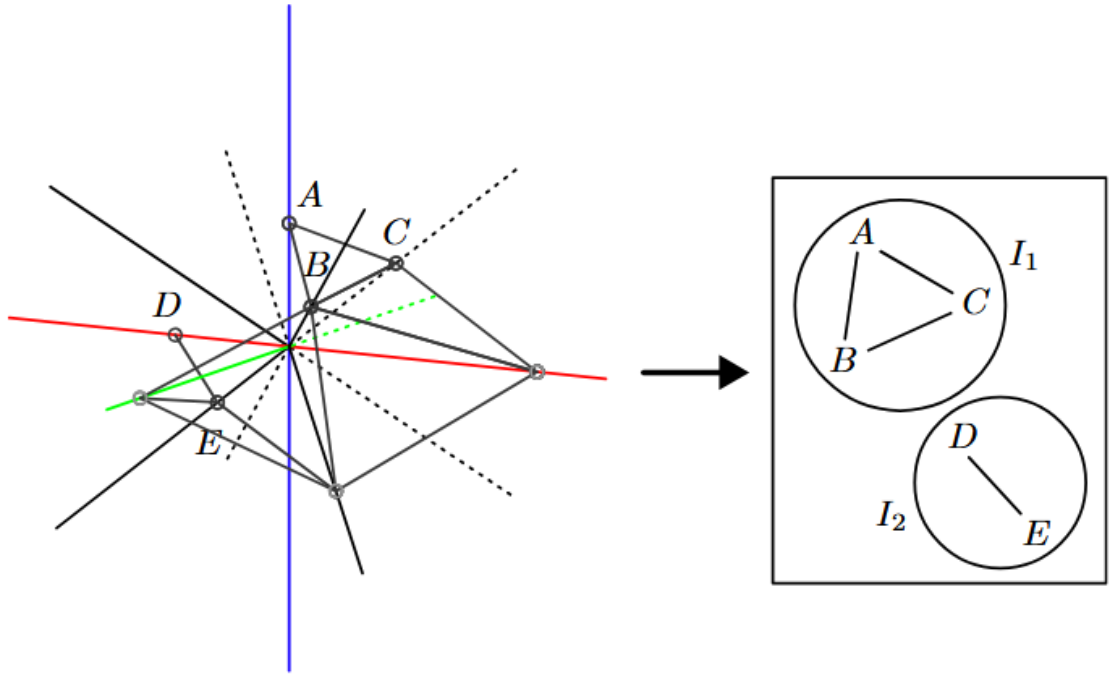


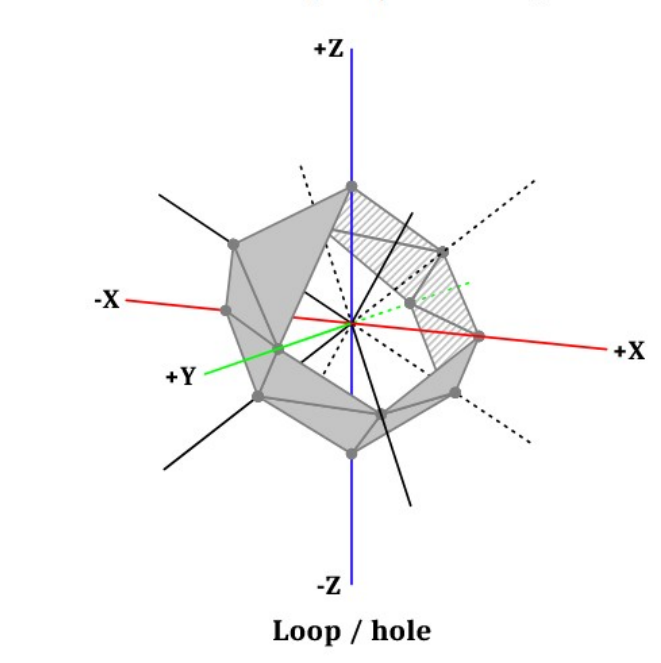
Figure 15: Example transformation from spatial representation into graph representation. As described, only nearby candidates participate (labelled $A \ldots E$), and only adjacent candidates may be connected togther. This results in the graph on the right, where it can be seen that the topology consists of two separate islands (labelled I_1 and I_2).

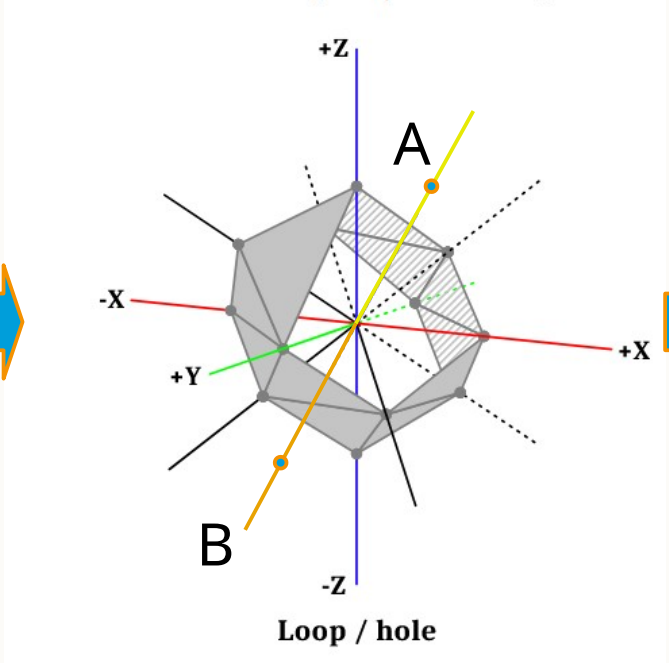
We store graphs as bitflag adjacency matrices

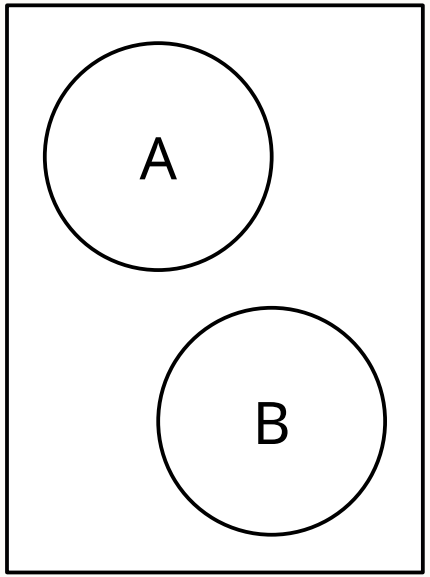
Most states relating to candidate vertices – such as which edges have them – can be stored as bitflags.
On the right is a 14x14 adjacency matrix where a 1 signifies and connection between candidate vertices.
This template is processed with bitwise AND operations to produce a graph of mergeable pairs of nearby intersections.

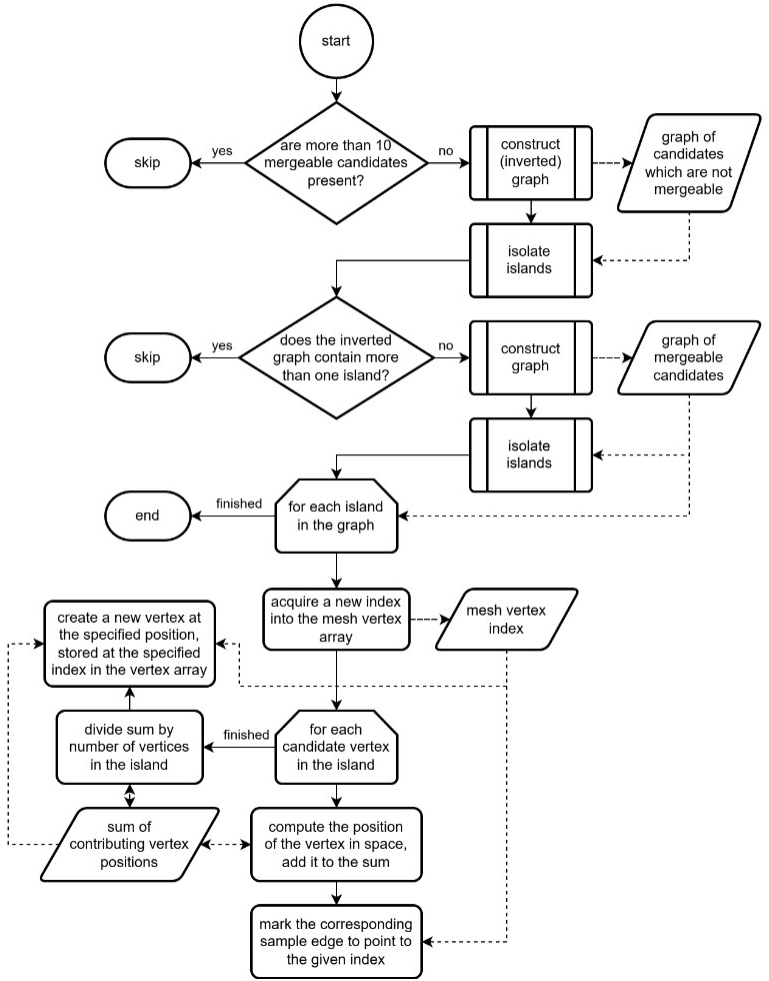
```
static constexpr EdgeFlags edge_neighbour_masks[14] =
{  // diag......perp..
    0b0001010101000000,     // PX
    0b0010101010000000,     // NX
    0b0000110011000000,     // PY
    0b0011001100000000,     // NY
    0b0000001111000000,     // PZ
    0b0011110000000000,     // NZ
    0b0000010110010101,     // PXPYPZ
    0b0000101001010110,     // NXPYPZ
    0b0001001001011001,     // PXNYPZ
    0b0010000110011010,     // NXNYPZ
    0b0001100001100101,     // PXPYNZ
    0b0010010010100110,     // NXPYNZ
    0b0001101000101001,     // PXNYNZ
   // diag......perp..
};
```

how do we identify the loop topology?







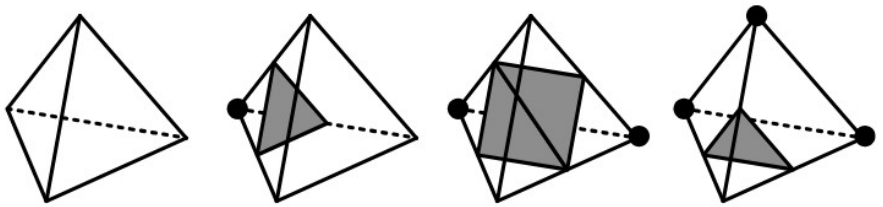

start
are more than 10 mergeable candidates present?
yes
skip
no
construct (inverted) graph
graph of candidates which are not mergeable
isolate islands
does the inverted graph contain more than one island?
yes
skip
no
construct graph
graph of mergeable candidates
isolate islands
for each island in the graph
finished
end
acquire a new index into the mesh vertex array
mesh vertex index
create a new vertex at the specified position, stored at the specified index in the vertex array
for each candidate vertex in the island
finished
divide sum by number of vertices in the island
compute the position of the vertex in space, add it to the sum
sum of contributing vertex positions
mark the corresponding sample edge to point to the given index

Algorithm 2: Vertex creation & clustering stage

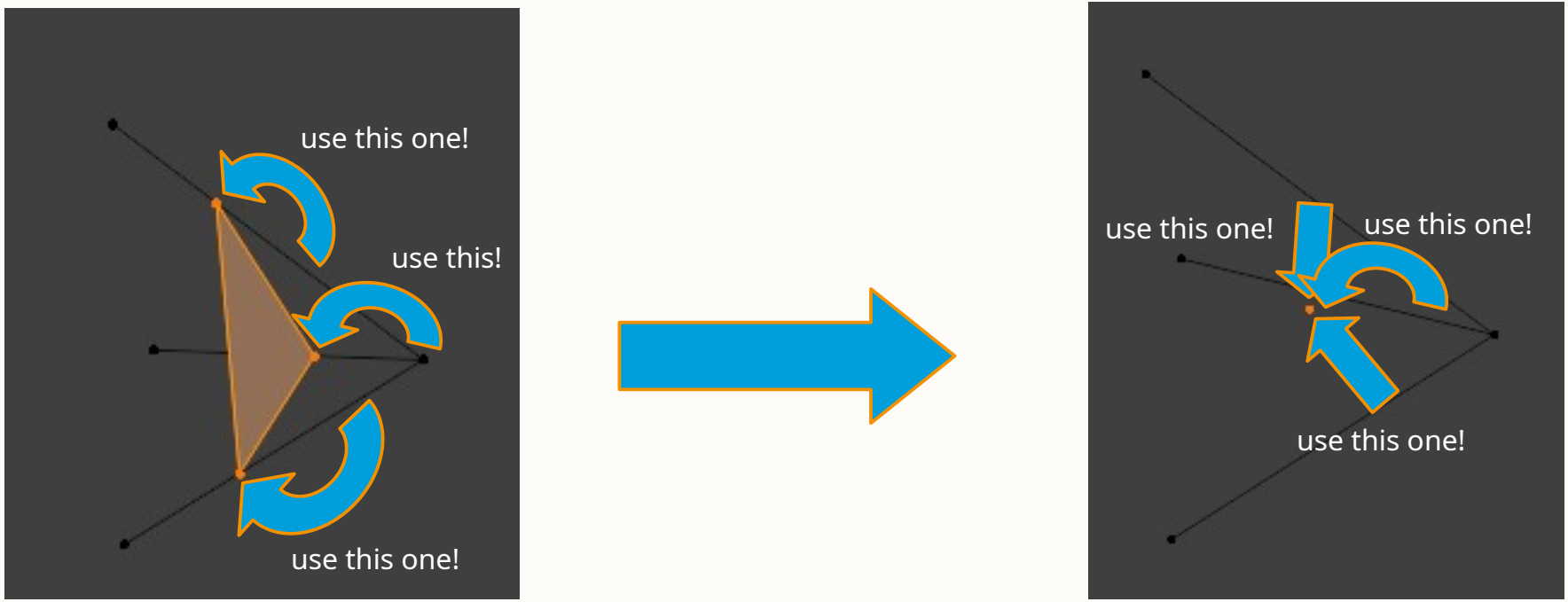
```
Let F be a smoothly defined scalar function on x, y, z
Let T be the threshold value for which the isosurface is being computed
Let A_v be the array of vertices in the generated mesh
for each sample point in the lattice do
    Let C equal the value of F retrieved from the current sample point
    Let S_1 ... S_14 equal the values of F retrieved from the neighbouring sample
     points in the lattice
    Let I be a series of Boolean flags representing isosurface intersections
     surrounding the current sample point
    for each neighbouring sample point do
        if sign(C − T) ≠ sign(Sn − T) and ‖C − T‖ < ‖Sn − T‖ then
            Set the flag in I corresponding to neighbour n
        end
    end
    Store the value of I at the current sample point
    Using I, construct a graph G representing which of these intersections can
     be merged together
    if G contains loops then
        Skip to the next sample point
    else
        Check for disconnected islands in G
        for each island in G do
            Create a single vertex R at the average position of all the isosurface
             intersections in the island
            Append R to the vertex array A_v and let V be its index within the
             array
            for each intersection in the island do
                Store V on the corresponding edge
            end
        end
    end
end
```

Algorithm 3: Geometry generation stage

```
Let A_v be the array of vertices in the generated mesh
Let A_t be the array of triangles in the generated mesh
for each cube in the sample lattice do
    Retrieve I from the sample point at the center of the cube
    if I has no flags set then
        Skip to the next sample cube
    end
    for each tetrahedron in the cube do
        Let B_1 ... B_4 be the states (inside or outside the isosurface) of the four
          corners of the tetrahedron
        Populate B_1 ... B_4 based on I and the state of the sample point at the
          center of the cube
        if all of B_1 ... B_4 are equal to one another then
            Skip to the next tetrahedron
        end
        Use the values of B_1 ... B_4 to select the triangulation scheme S from a
          pre-defined list
        Let V_1 ... V_4 be the indices into A_v, retrieved from the relevant edges
          as described by S
        Based on S, construct triangles R_1 ... R_k, where k may be 1 or 2, using
          V_1 ... V_4
        for each triangle R_n in R_1 ... R_k do
            if R_n contains one or more duplicated index then
                Discard R_n
            else
                Append R_n to the triangle array A_t
            end
        end
    end
end
```



What if instead of three separate vertices (one for each intersection)...



multiple edges reference the same vertex, as a result of merging...? **Ignore that triangle!**

Optimisations

- Pre-computed relative position offset table
- Proximity flags (I) are stored to allow skipping a large number of tetrahedra (and memory accesses) in the geometry pass (Bajaja et al., 1996)
- Vertex normals calculated by average instead of trying to sample surface gradient (unlike Treece, Prager, and Gee (1999))

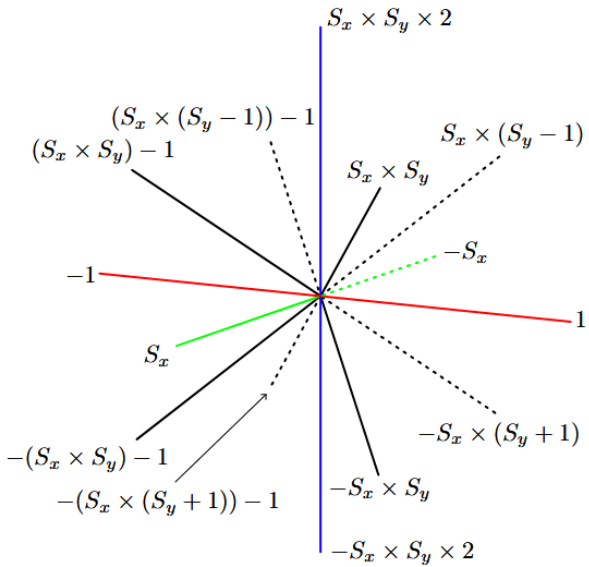


Figure 17: Visual depiction of the table of sample index offsets, for the body-centered tetrahedron arrangement, on even slices of the Z axis. S_x is defined to be the number of samples present in the array on the X axis, while S_y is defined likewise for the Y axis.

Parallelisation

- Sampling pass was parallelised, since testing found that this is the most expensive area of the algorithm and is down to the complexity of the user's sampling function.
- Vertex pass *could* be parallelised, but its cost is insignificant compared to the sampling pass, and independent threads generating separate arrays of vertices would need to be merged after the fact, potentially at significant extra time cost.
- Dividing the sample volume into chunks independently.

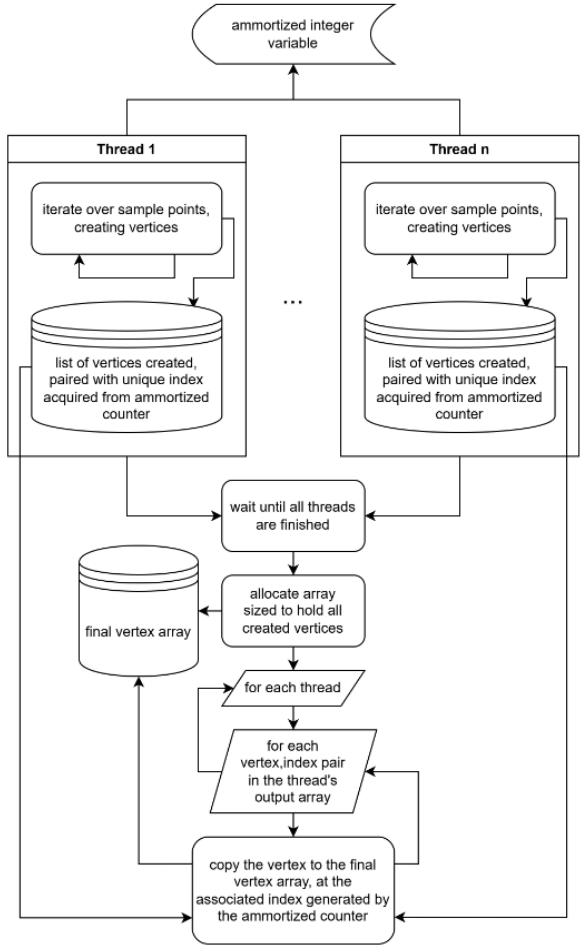


Figure 20: Example algorithm for how the vertex creation pass could be parallelised. The time spent within critical sections - using the ammortized counter, which is a synchronised, one-at-a-time multi-threaded resource - is minimised, and vertices are merged afterwards at additional cost. The use of an ammortized counter to number vertices guarantees each vertex is uniquely identified by its index, preserving the existing behaviour of the geometry generation pass.

Tetrahedral Arrangements

- We present an alternative version of the ‘simple cubic’ arrangement which does not require mirroring its shape on alternate cubes.
- Simplifies implementation, and makes all tetrahedra congruent, at the cost of less regular-shaped tetrahedra (and directional bias).

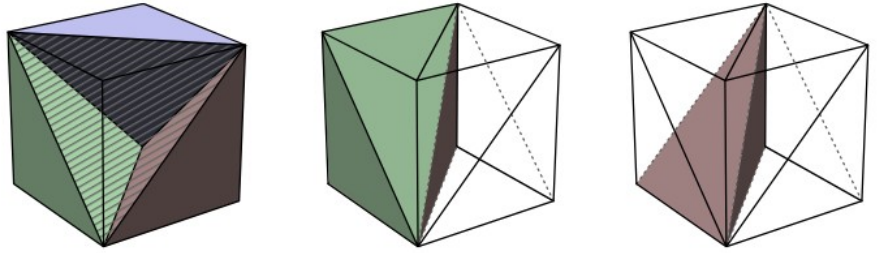


Figure 19: Left: Original simple cubic arrangement (shown again for reference), containing 5 tetrahedra in total, with 3 coloured solid.
Center: Our simple cubic arrangement, containing 6 tetrahedra in total, with 2 coloured solid.
Right: Same as center, but with only 1 tetrahedron coloured solid to clarify the structure.

Other Artefact Notes

- I built a 3D test environment using OpenGL and ImGui where you can play with parameters in real-time
- I built an octree-accelerated mesh nearest-point SDF algorithm for use with the Bunny benchmark
- I also built a tool for generating GIFs for use in the questionnaire

END OF ARTEFACT

What are we Evaluating, and how?

- The main points of comparison between different approaches are **generation time**, **triangle count**, and **triangle/shading quality**
- Generation time and triangle count are both quantitatively measured using benchmarks (as well as triangle aspect ratio)
- Shading quality is measured statistically via a questionnaire investigating ‘visual preference’ between samples

What Parameters are Changing?

There are 18 combinations of test parameters in total:

- *Sample object* (3) – sphere, asteroid, or Stanford Bunny
- *Tetrahedron arrangement* (2) – body-centered diamond lattice, or simple cubic
- *Clustering method* (3) – no clustering, RMT ‘integrated’ clustering, or edge-collapse ‘post-processed’ merging

For example, this animation shows the Bunny sample object, using the the BCDL tetrahedron arrangement, and the integrated clustering method

6 Appendix A - Test Parameter Configurations

Below is a list of configurations of test parameters used in the benchmark and questionnaire data gathering. Each is accompanied by an 4-letter name used to uniquely and anonymously identify the configuration.

- Sphere object; body-centered arrangement; no clustering - `ss_bu`
- Sphere object; body-centered arrangement; integrated clustering - `ss_bi`
- Sphere object; body-centered arrangement; post-processed clustering - `ss_bp`
- Sphere object; simple arrangement; no clustering - `ss_su`
- Sphere object; simple arrangement; integrated clustering - `ss_si`
- Sphere object; simple arrangement; post-processed clustering - `ss_sp`
- Asteroid object; body-centered arrangement; no clustering - `as_bu`
- Asteroid object; body-centered arrangement; integrated clustering - `as_bi`
- Asteroid object; body-centered arrangement; post-processed clustering - `as_bp`
- Asteroid object; simple arrangement; no clustering - `as_su`
- Asteroid object; simple arrangement; integrated clustering - `as_si`
- Asteroid object; simple arrangement; post-processed clustering - `as_sp`
- Bunny object; body-centered arrangement; no clustering - `bs_bu`
- Bunny object; body-centered arrangement; integrated clustering - `bs_bi`
- Bunny object; body-centered arrangement; post-processed clustering - `bs_bp`
- Bunny object; simple arrangement; no clustering - `bs_su`
- Bunny object; simple arrangement; integrated clustering - `bs_si`
- Bunny object; simple arrangement; post-processed clustering - `bs_sp`

Benchmarking

What

- Time taken (and where it was spent)
- Mean aspect ratio of triangles
- Number of triangles

Where

- HP Z2 G8 Tower Workstation Desktop PC
- 11th Gen Intel Core i7-11700 CPU @2.5GHz
- 32GB of 3200MT/s DDR4 RAM

Each is benchmark executed 100 times, and results are averaged.

Questionnaire

- Participants compare a series of pairs of animations
- One parameter is varied between animations (different sample objects are not compared), to isolate variables
- Participants choose either *'Prefer Left'*, *'No Preference'*, or '*Prefer Right'* for each question
- There are 27 questions in total
- There were 18 responses, and volunteer sampling was used for selection

5

Which of these two animations do you prefer? *

- ○ Prefer left
- ○ No preference
- ○ Prefer right

Making the Animations

- GIFs were made with each configuration of test parameters using the artefact
- These were combined using FFmpeg and upscaled
- Video files are named based on the animations they compare, using non-indicative names

Questionnaire Wistful Thinking

- Ideally each comparison would be repeated in the opposite order (i.e. swapping left with right and running the question again)
- Ideally we would have some control questions (i.e. comparing two of the same)
- Length of questionnaire is important – too long and people stop caring
- More participants is always positive

Questionnaire Analysis

- Groups of comparisons (e.g. 3 questions comparing different merging methods while other parameters remain constant) are collected together
- An overall preference factor is calculated for each configuration of test parameters
- All versions of this comparison group for different sample objects are combined using an arithmetic mean
- Overall preference factor for specific test parameter (e.g. integrated merging) is calculated by adding factors from different groups

END OF EVALUATION METHODOLOGY

The Results

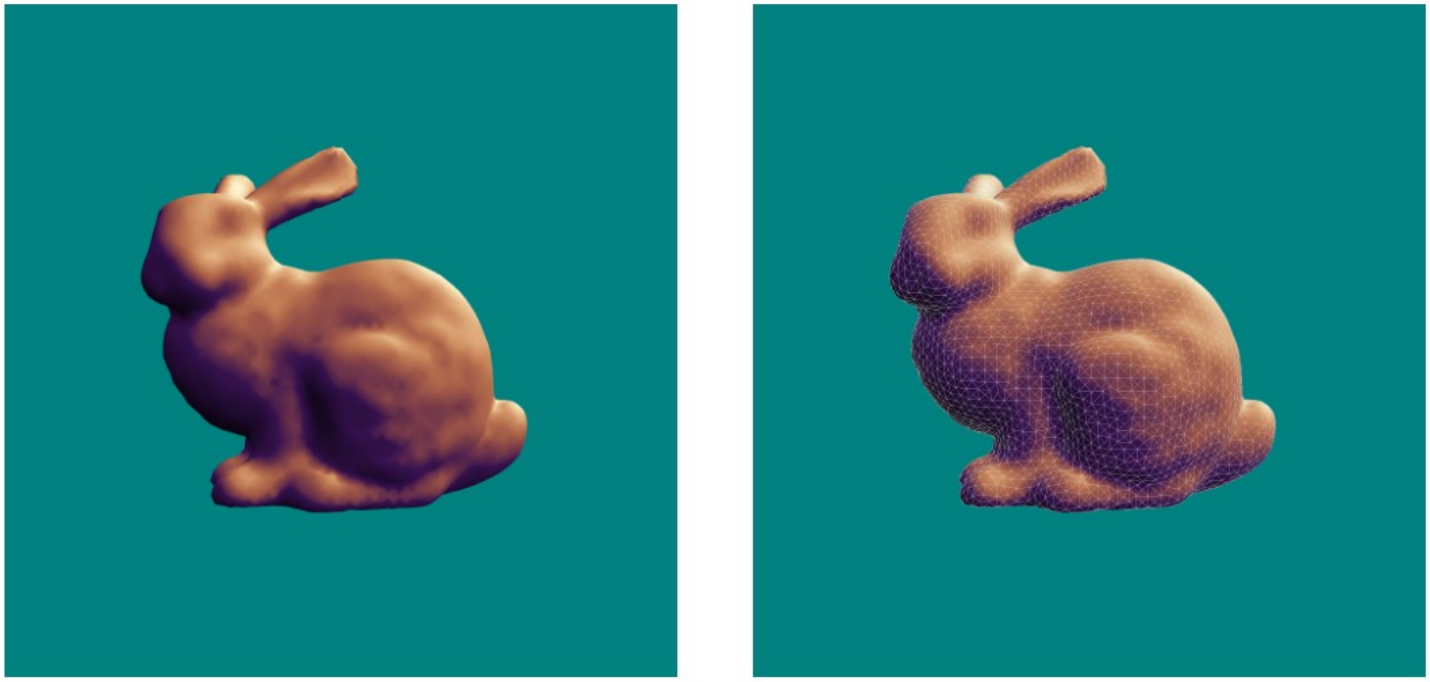
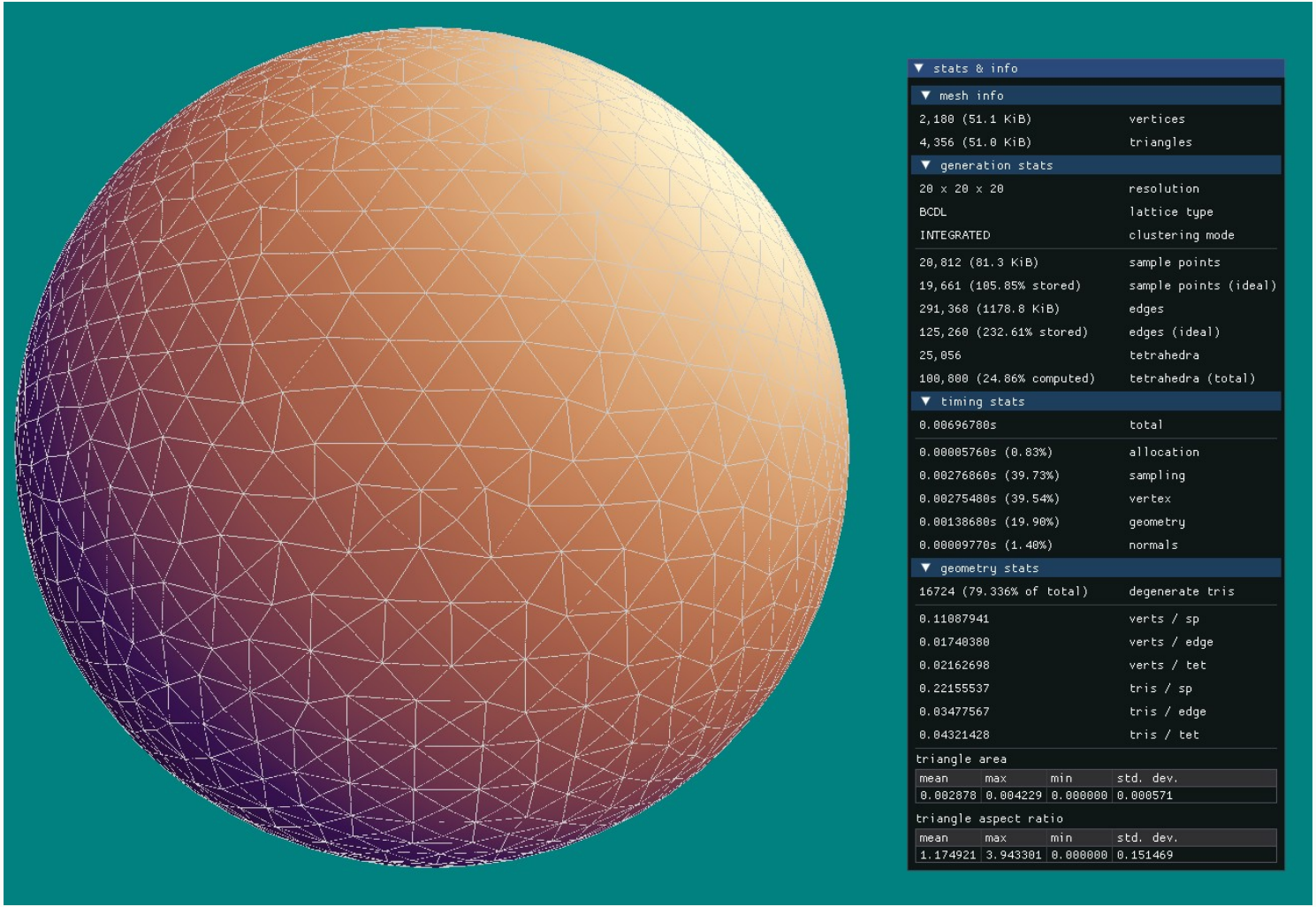


Figure 21: Images of mesh output for the Stanford Bunny, using a body-centered arrangement with the integrated clustering algorithm, with the wireframe shown on the right.

The Results

Figure 22: Images of mesh output for the sphere with a low sample resolution, using a body-centered arrangement with the integrated clustering algorithm. Flat shading is shown in the center, with the wireframe shown on the right.

Overall – highly successful in the underlying goal!

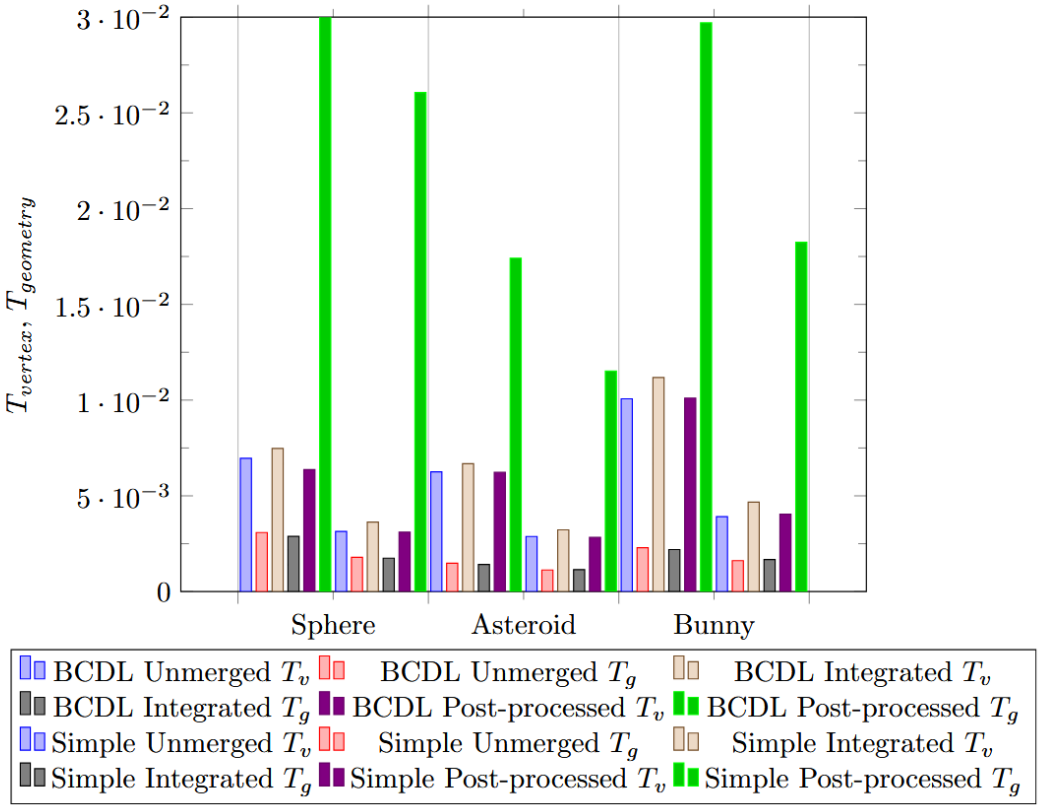

▼ stats & info
▼ mesh info
2,180 (51.1 KiB) vertices
4,356 (51.0 KiB) triangles
▼ generation stats
20 x 20 x 20 resolution
BCDL lattice type
INTEGRATED clustering mode
20,812 (81.3 KiB) sample points
19,661 (105.85% stored) sample points (ideal)
291,368 (1178.8 KiB) edges
125,260 (232.61% stored) edges (ideal)
25,056 tetrahedra
100,800 (24.86% computed) tetrahedra (total)
▼ timing stats
0.00696780s total
0.00005760s (0.83%) allocation
0.00276860s (39.73%) sampling
0.00275480s (39.54%) vertex
0.00138680s (19.90%) geometry
0.00009770s (1.40%) normals
▼ geometry stats
16724 (79.336% of total) degenerate tris
0.11087941 verts / sp
0.01740380 verts / edge
0.02162698 verts / tet
0.22155537 tris / sp
0.03477567 tris / edge
0.04321428 tris / tet
triangle area
mean max min std. dev.
0.002878 0.004229 0.000000 0.000571
triangle aspect ratio
mean max min std. dev.
1.174921 3.943301 0.000000 0.151469

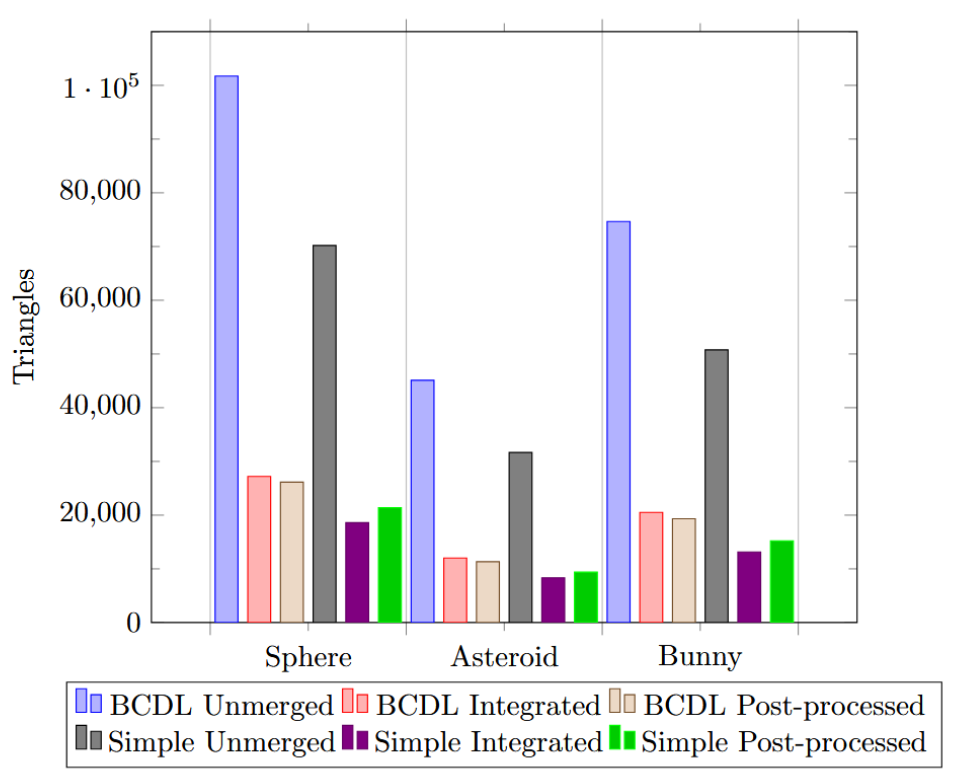
Benchmark Results

ID	T_{vertex}	$T_{geometry}$	$N_{vertices}$	$N_{triangles}$	AR_{mean}	AR_{sd}	$F_{discard}$
as_bu	0.00625	0.00144	22550	45096	5.81	78.56	0.000%
as_bi	0.00668	0.00142	6012	12020	1.16	0.15	79.47%
as_bp	0.00623	0.0174	5665	11326	1.23	0.21	74.88%
as_su	0.00288	0.00113	15840	31676	6.53	106.34	0.000%
as_si	0.00323	0.00115	4172	8340	1.33	0.28	79.70%
as_sp	0.00283	0.0115	4697	9390	1.39	0.32	70.36%

- Regularisation significantly improves triangle aspect ratio, and massively reduces triangle count
- Post-processed merging is much more expensive time-wise compared to integrated merging

Benchmark Results





Benchmark Results

- Integrated merging is sometimes cheaper than no merging
- Simple cubic is generally faster and produces less triangles – however, there are also fewer tetrahedra involved per unit volume!
- Post-processed merging produces slightly more equilateral (better quality) triangles than integrated merging, but is much slower and still produces more triangles overall

Questionnaire Results

B arrangement			S arrangement			U clustering		I clustering		P clustering	
U	I	P	U	I	P	S	B	S	B	S	B
0.105	0.419	0.358	0.154	0.562	0.235	0.185	0.667	0.500	0.370	0.352	0.519

	Body-centered	Simple cubic	Mean
Unmerged	0.772	0.339	0.556
Integrated	0.790	1.062	0.926
Post-processed	0.877	0.586	0.731
Mean	0.813	0.663	

Questionnaire Results

- Body-centered is generally preferred as an arrangement
- Integrated merging is generally preferred for merging
- Both congruent with literature
- However, despite this, the singular most popular configuration was *simple cubic* integrated
- Possibly due to a limited sample size and only three sample models – simple cubic may be best suited for highly irregular models such as the Bunny

END OF RESULTS

Key Conclusions

- Regularisation of some sort is super, super important, both in terms of mesh rendering/storage cost, as well as shading quality and visual preference
- The preferred tetrahedral tessellation scheme may vary depending on the use case
- Integrated merging such as RMT outpaces other clustering techniques by almost a factor of 10
- Probably more study with a bigger sample size and more varied models is needed

Improvements to the Algorithm

- Memory usage gets prohibitive for large sample volumes; edge data is technically stored twice to speed up retrieval
- Try a different IE approach entirely, since preserving fine surface detail isn't super important
- Take advantage of gradient info about the sample function
- My merging method isn't perfect!

20 x 20 x 20	resolution
BCDL	lattice type
INTEGRATED	clustering mode
20,812 (81.3 KiB)	sample points
19,661 (105.85% stored)	sample points (ideal)
291,368 (1178.8 KiB)	edges
125,260 (232.61% stored)	edges (ideal)

Other Future Work

- A closer look at visual preference with *flat* shaded meshes – with regard to ASTRONEER and its art style
- Development of novel isosurface extraction techniques specifically suited for videogames, particularly chunk-based or incremental generation
- Greater exploration of dual contouring and marching front algorithms compared to RMT

Demo Time!

Costen (2026) – also they are friends, please do not separate

END OF EVERYTHING