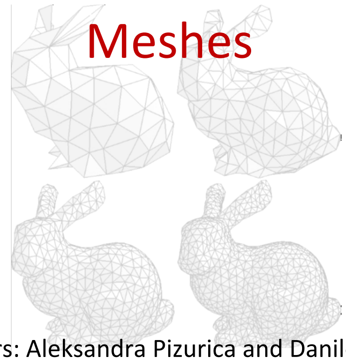
FACULTY OF ENGINEERING AND ARCHITECTURE

E016712: Computer Graphics

Meshes



Lecturers: Aleksandra Pizurica and Danilo Babin

GHENT UNIVERSITY

Meshes for Surface Rendering

Represent surfaces created from images or geometric models

Visualization of surfaces is known as surface rendering

Overview

- Mesh generation from:
  - Implicitly defined surfaces
  - Explicitly defined surfaces
  - Mapping from images
- Mesh subdivision strategies
  - Butterfly
  - Loop
  - Catmull-Clark
  - Wavelets

Volumetric data

Choose a fixed point

$$\mathbf{c}_0 = (x_0, y_0, z_0)$$

An arbitrary point within the volume can be represented as

$$F(\mathbf{p}) = F(x, y, z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}.$$

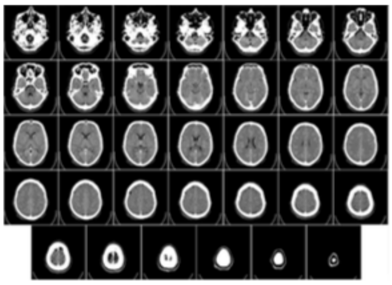
A major problem is visualization on a 2D display

Slicing to Visualize Volumetric Data

Examples: CT scan, MRI scan

Conventional approach is to slice the data

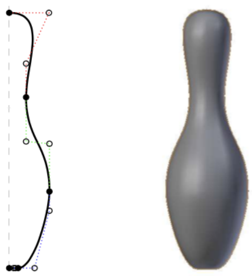
- often visualized in short video sequences showing slices in a rapid succession



Surface Approximations

For arbitrary surfaces exact mathematical descriptions often become cumbersome

Representations based on Bézier curves and splines are popular in CAD applications, but too complex for real-time interactive applications



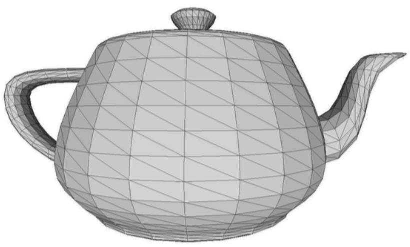
A bowling pin modeled using a Bézier curve. The surface on the right is obtained by spinning the curve around a fixed center axis.
Example from the PhD thesis Jonas El Sayeh Khalil, Ghent University 2018.

Introduction to Meshes

- In computer graphics, we do not draw surfaces that are truly curved.
- Instead, each surface consists of many small polygons connected in a mesh.
- This is also called a wireframe.
- If we fill the polygons in the mesh, then the surface looks solid.

Digital Surfaces - 3D Mesh

Polygon mesh



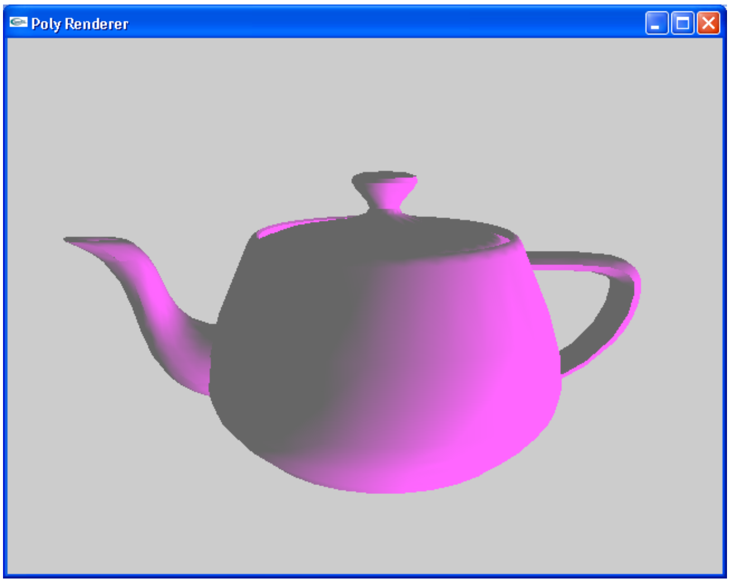
Mesh representation involves

- VERTICES – geometry information
- FACES – connectivity information

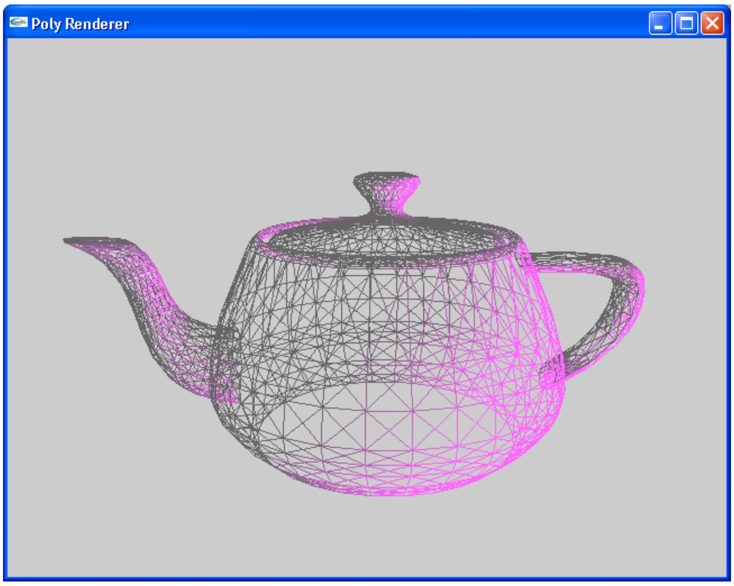
Triangular meshes

- Most often encountered in real-time rendering
- Described by a list of vertices and a list of triangles
- Any mesh can be transformed to a triangular mesh

Solid Model

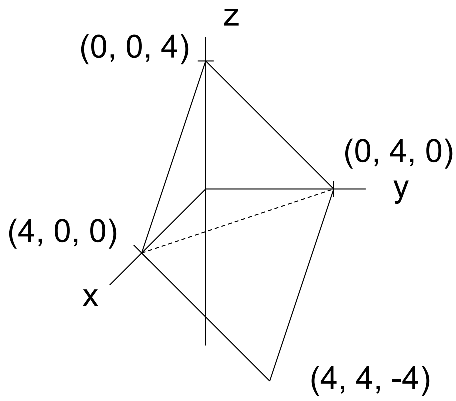


Wireframe Model



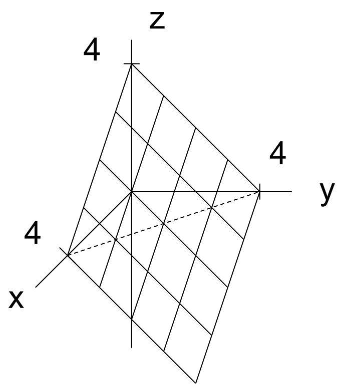
Surfaces Defined Explicitly

- Let $z = f(x, y)$ be a function of two variables.
- This function defines a surface over the xy-plane.
  - For every point (x, y) in the plane, the function produces a z-coordinate, giving a point (x, y, z) in space.
- Let $z = 4 - x - y$.



Creating a Rectangular Mesh

- It is clear that we can create a rectangular mesh for a surface represented explicitly as $z = f(x, y)$ over a rectangular region $[a, b] \times [c, d]$.
- Simply subdivide the range [a, b] of values of x into $a = x_0, x_1, \ldots, x_n = b$.
- Similarly, subdivide the range [c, d] of values of y into $c = y_0, y_1, \ldots, y_m = d$.



Normals

- In order for the lighting effects to be computed properly, we must also create a unit normal vector at each grid point.
- This normal should be perpendicular to the tangent plane.
- To compute it, we take the cross product of two vectors lying in the tangent plane.

Isosurfaces

Isolines connect all points of a 3D volume, which have specific values

- Examples: isophyses on an elevation map; isobars on a weather map

Volumetric data can be visualized using isosurfaces

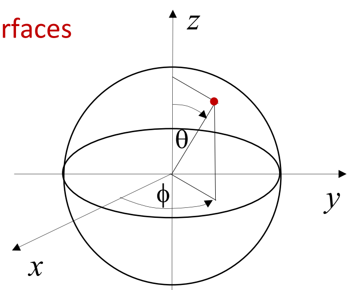
- Choosing a set of points on a sphere

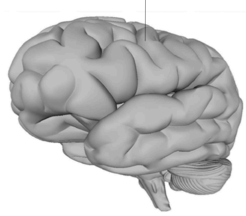
$$F(x, y, z) = r$$

- Such a surface is in fact a 2D surface

$$(\phi, \theta) \mapsto (x, y, z)$$

For a fixed radius r





Surfaces Defined Implicitly

- Let $F(x, y, z) = 0$ be an equation that implicitly defines a 2-dimensional surface in 3-dimensional space.
- For example,
  - $x^2 + y^2 + z^2 - 1 = 0$ defines a sphere.
  - $x^2 + y^2 - z^2 = 0$ defines a cone.

Parameterizing the Surface

- Often the surface can be parameterized in terms of two parameters u and v, based on some intrinsic properties of the surface.
- Let x, y, z be given in terms of u and v, with $0 \le u \le 1$, $0 \le v \le 1$.
  - $x = x(u, v)$, $y = y(u, v)$, $z = z(u, v)$.
- Then $P(u, v) = (x, y, z)$ is a point on the surface.

Finding Normal Vectors

- At a point P(u, v),
  - The tangent to the u-contour is given by $\partial P/\partial u$.
  - The tangent to the v-contour is given by $\partial P/\partial v$.
  - Thus, the unit normal to the surface is $\mathbf{n} = (\partial P/\partial u) \times (\partial P/\partial v)$, normalized.

- Directions of normals:
  - This proceed is as likely to produce vectors that point "inward" as it is to produce vectors that point "outward."
  - If the vectors point inward, then attach a negative sign to each component to reverse the direction.

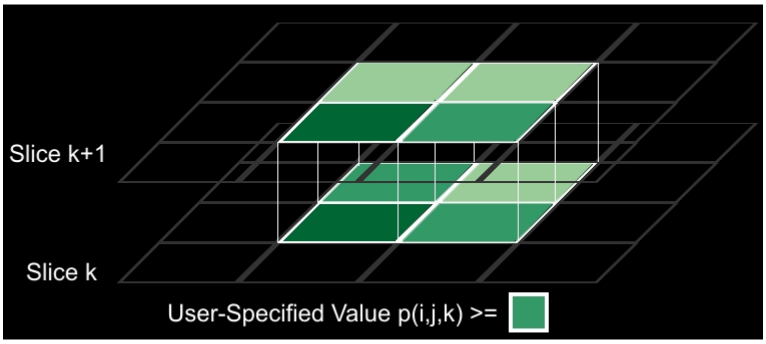
Marching Cubes Algorithm

Consists of 3 basic steps:

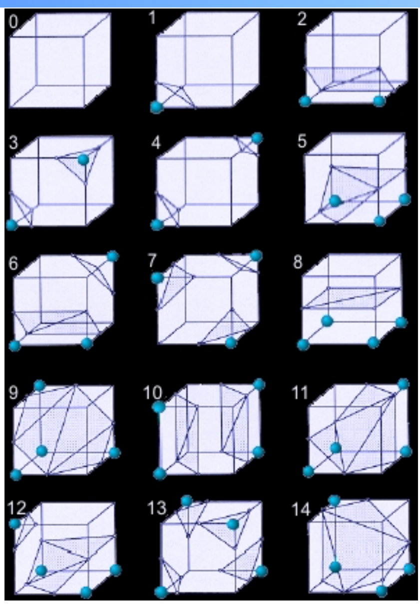
1. Locate the surface corresponding to a user-specified value.
2. Create triangles.
3. Calculate normals to the surface at each vertex.

Step 1: definition of foreground pixels

- Uses a moving window of eight pixels (4 from each 2 adjacent slices
- Considers only pixels with values above a user specified threshold value.



Step 2 : Triangulation



- For each cube, we have 8 vertices with 2 possible states each (inside or outside).
- This gives us 2^8 possible patterns = 256 cases.
- Enumerate cases to create a LUT
- Use symmetries to reduce problem from 256 to 15 cases.

Step 3 : Surface normals

- To calculate surface normal, we need to determine gradient vector, G (derivative of the density function).
- We first estimate the gradient vectors at the vertices and interpolate the gradient at the intersection.
- The gradient at cube vertex (i , j, k), is estimated using central differences along the three coordinate axes by:

D (i, j, k) is the density at pixel (i, j) in slice k.

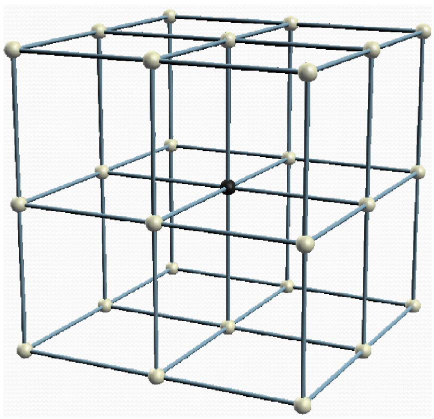
Δx, Δy, Δz are lengths of the cube edges

$$G_x(i, j, k) = \frac{D(i+1, j, k) - D(i-1, j, k)}{\Delta x}$$

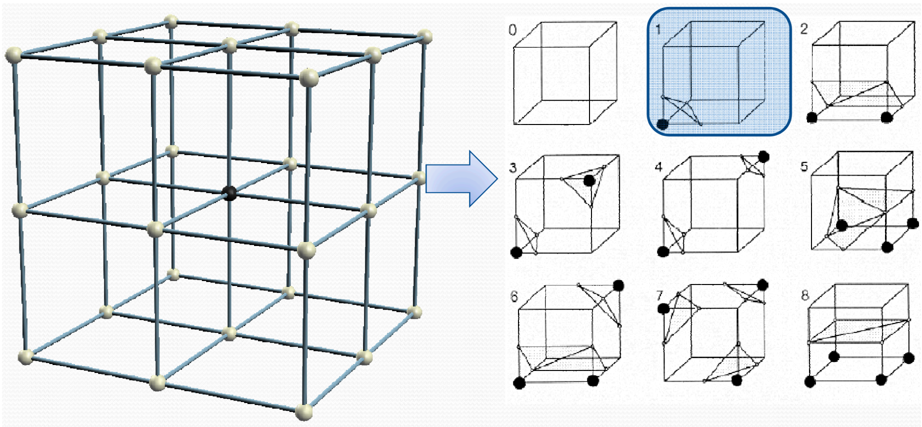
$$G_y(i, j, k) = \frac{D(i, j+1, k) - D(i, j-1, k)}{\Delta y}$$

$$G_z(i, j, k) = \frac{D(i, j, k+1) - D(i, j, k-1)}{\Delta z}$$

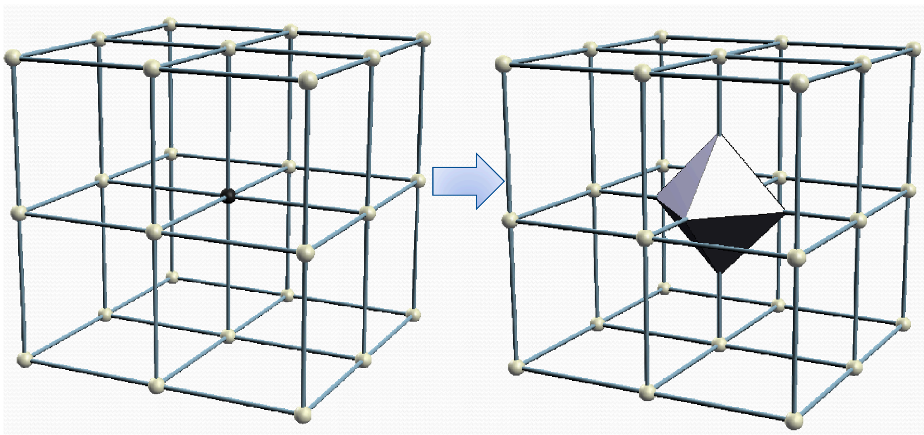
Marching Cubes: Example



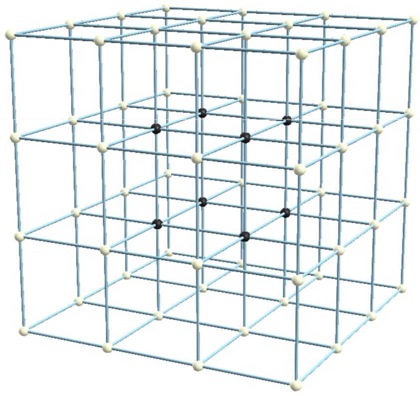
Marching Cubes: Example



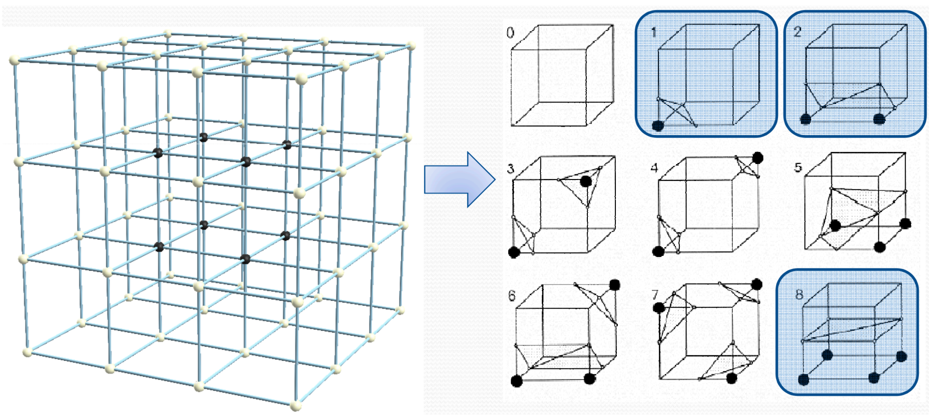
Marching Cubes: Example



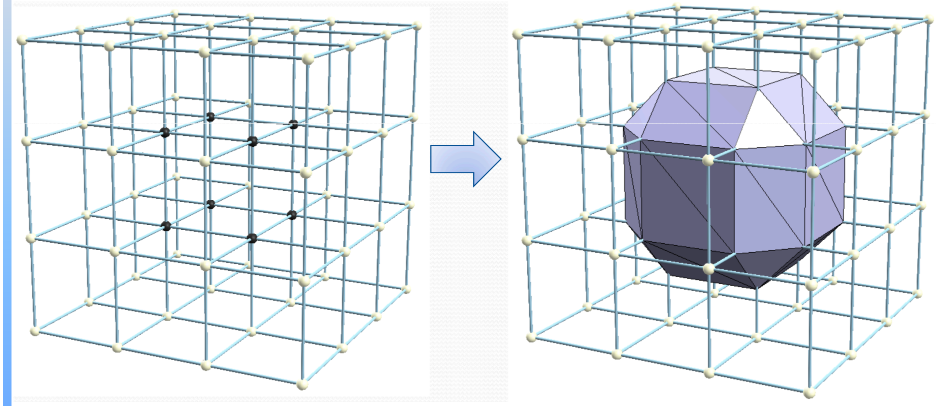
Marching Cubes: Example



Marching Cubes: Example



Marching Cubes: Example



Distortion types

In general, distortion of a digital representation is related to

- resolution (determined by sampling density)
- quality of samples (determined by quantization)

Take for example quality of image coding:

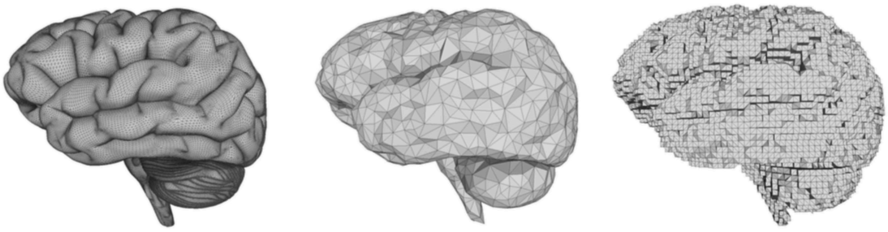
original low resolution low quantization

Distortion types

In general, distortion of a digital representation is related to

- resolution (determined by sampling density)
- quality of samples (determined by quantization)

Quality of mesh coding:



Original	Low resolution	Low quantization
294 012 vertices	3941 vertices	294 012 vertices

Example from the PhD thesis Jonas El Sayeh Khalil, Ghent University 2018.

Need for Scalable Representations

Today, models of millions or even billions of vertices are no exception!

Example: Digital Michelangelo Project: a 3D mesh representation using billions of vertices

https://graphics.stanford.edu/projects/mich/

Need for Scalable Representations

Interactive visualization of models with millions or more vertices is not evident.

We typically need a storage of 30 bytes/vertex

- each vertex stored with 3 x 2-byte coordinates
- each triangle stored with 3 x 4-byte indices into the list of vertices
- 2 triangles per vertex (on average)

Example: NVIDIA GTX 1080 with 8 GB memory

- Can render models of maximally 250 – 300 millions of vertices

	GPU memory	Largest mesh	
NV GTX 1080	8 GB	267×10^6 vertices	
NV GF 940M	4 GB	133×10^6 vertices	
iPhone X	3 GB	100×10^6 vertices	(shared with the CPU!)
LG Nexus 5X	2 GB	67×10^6 vertices	(shared with the CPU!)

Example from the PhD thesis Jonas El Sayeh Khalil, Ghent University 2018.

Need for Scalable Representations

Rendering at 60 frames per second (fps):

- Approximately 16.7ms are available between rendering one frame and the next
- If models are not available in time: popping effect (geometry or texture data is not visible instantaneously)

Example: GPU NVIDIA GTX 1080 offers a bandwidth of 320 GB/s or 320 MB/ms → 10 000 000 of vertices (and their associated triangles) can be streamed per ms

	Bandwidth	Geometry per second	
NV GTX 1080	320 GB/s	10×10^6 vertices/ms	(GPU upload)
NV GF 940M	14.4 GB/s	480×10^3 vertices/ms	(GPU upload)
SSD	500 MB/s	16.7×10^3 vertices/ms	(RAM upload)
WiFi	100 Mbps	417 vertices/ms	(LAN download)
Coax	30 Mbps	125 vertices/ms	(WAN download)

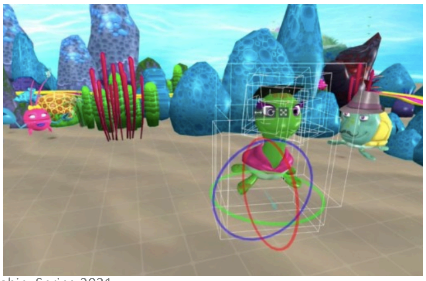
Example from the PhD thesis Jonas El Sayeh Khalil, Ghent University 2018.

Need for Scalable Representations

The limitations in the examples above consider virtual environments with only a single model to be viewed

When creating interactive virtual worlds parts of the entire scene need to be stored

- Depending on the scene complexity, tens or hundreds of objects need to be available simultaneously
- Other attributes: diffuse and specular color values, surface reflectivity, transparency,…



Mesh compression

Compression is needed for efficient storage and transmission

Compression techniques exploit **correlations**, local **similarities** and **predictability** to store the data more efficiently

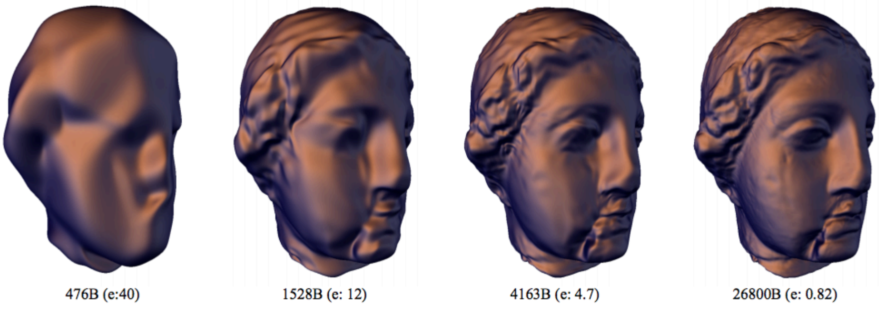
The performance of a **codec** (coder + decoder) is expressed as a **bit rate**; for meshes: the required amount of **bits per vertex** (bpv)

State-of-the-art single-rate compression: 1.6 bpv

- A billion vertex model requires 200 MB of storage space
  - Takes 400 ms to fetch from a SSD and 16s to download from a local server

To reduce popping effects, a conventional approach is to use several **Levels of Detail** (**LoDs**)

Progressive mesh compression



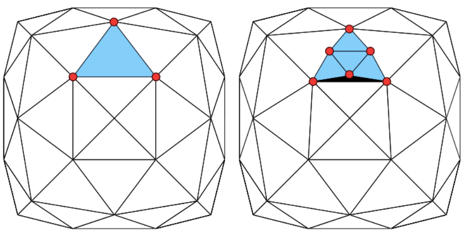
Partial bit-stream reconstructions from a progressive encoding of the Venus head model. File sizes are given in bytes and relative L2 reconstruction error in multiples of 10^{-4}. The rightmost reconstruction is indistinguishable from the original.

A. Khodakovsky, P. Schröder and W. Sweldons: Progressive Geometry Comresssion, SIGGRAPH 2000, pp. 271-278.

Subdivision surfaces

Subdivision surfaces are limit surfaces that result from recursive refinement of polygonal base meshes.

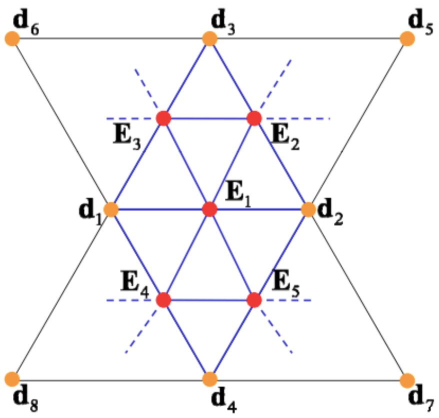
A subdivision step refines a **submesh** to a **supermesh** by inserting vertices.



Different subdivision schemes exist

- **Butterfly** subdivision
- **Loop** (best for triangular meshes)
- **Catmull-Clark** (best for quad meshes)

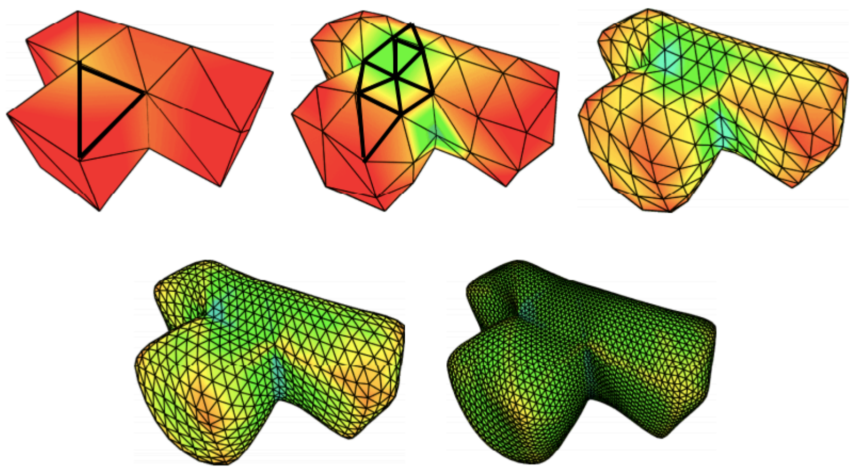
Subdivision surfaces: Butterfly subdivision



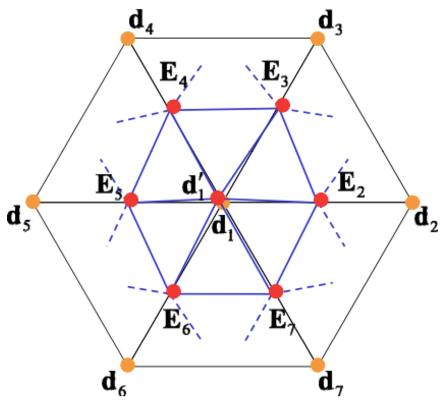
$$\mathbf{E}_1 = \frac{1}{2}(\mathbf{d}_1 + \mathbf{d}_2) + \omega(\mathbf{d}_3 + \mathbf{d}_4) - \frac{\omega}{2}(\mathbf{d}_5 + \mathbf{d}_6 + \mathbf{d}_7 + \mathbf{d}_8)$$

$$\mathbf{d}_i' = \mathbf{d}_i$$

Subdivision surfaces: Butterfly subdivision



Subdivision surfaces: Loop subdivision

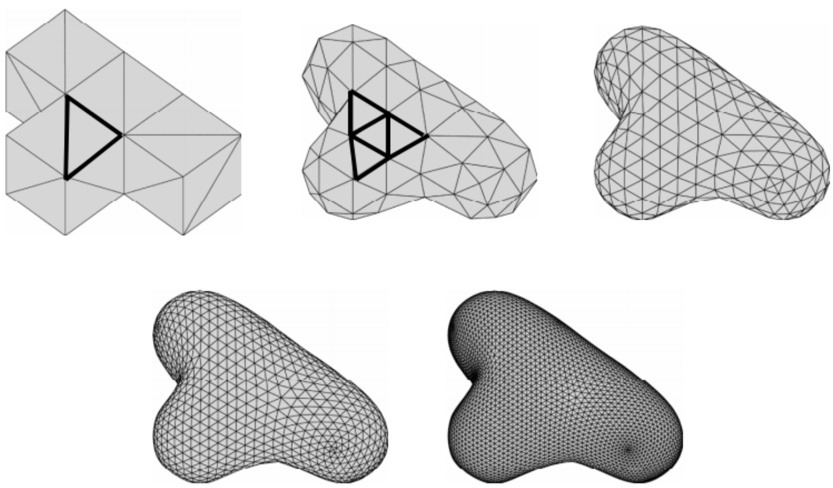


$$E_i = \frac{3}{8}(d_1 + d_i) + \frac{1}{8}(d_{i-1} + d_{i+1})$$

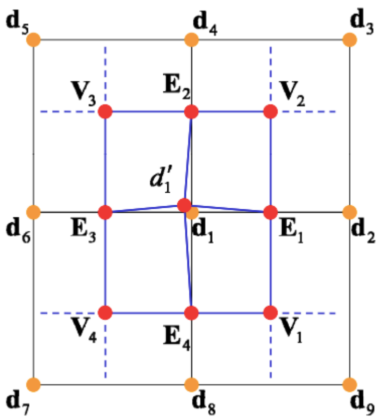
$$\mathbf{d}_1' = \alpha_n \mathbf{d}_1 + \frac{(1-\alpha_n)}{n} \sum_{j=2}^{n+1} \mathbf{d}_j$$

$$\alpha_n = \frac{3}{8} + \left(\frac{3}{8} + \frac{1}{4} \cos \frac{2\pi}{n} \right)^2$$

Subdivision surfaces: Loop subdivision



Subdivision surfaces: Catmull-Clarck



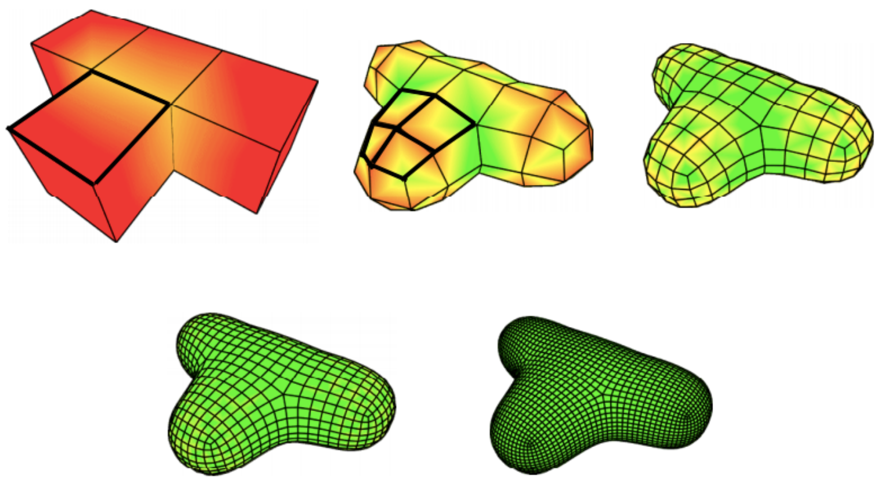
$$\mathbf{V}_2 = \frac{1}{n} \times \sum_{j=1}^{n} \mathbf{d}_j$$

$$\mathbf{E}_i = \frac{1}{4}\left(\mathbf{d}_1 + \mathbf{d}_{2i} + \mathbf{V}_i + \mathbf{V}_{i+1}\right)$$

$$\mathbf{d}_1' = \frac{(n-3)}{n}\mathbf{d}_1 + \frac{2}{n}\mathbf{R} + \frac{1}{n}\mathbf{S}$$

$$\mathbf{R} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{E}_i \quad \mathbf{S} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{V}_i$$

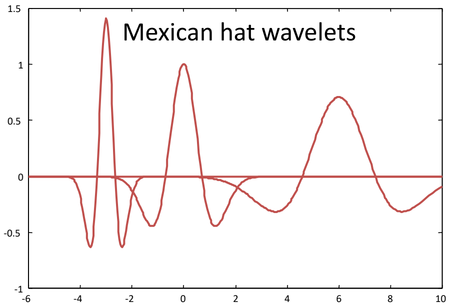
Subdivision surfaces: Catmull-Clarck



The wavelets concept- localized waves

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\,\psi\left(\frac{t-b}{a}\right)$$

Wavelet family:
shifts and dilations of the
mother wavelet $\psi(t)$



- Continuous wavelet transform: correlate signal with wavelets to reveal structures of different sizes
- Main idea: **analyze according to scale**
  (see the forest and the trees!)

Continuous wavelet transform (CWT)…

- The continuous wavelet transform (CWT) of a signal $f(x)$ is

$$\mathcal{W}f(a,b) = \frac{1}{\sqrt{a}}\int_{-\infty}^{\infty} f(x)\psi^*\left(\frac{x-b}{a}\right)dx \;=\; \langle f, \psi_{a,b}\rangle$$

- $\psi^*(x)$ denotes the complex conjugate of $\psi^*(x)$
- The inverse transform:

$$f(x) = \left(\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathcal{W}f(a,b)\psi_{a,b}(x)dadb/a^2\right)/C_\psi$$

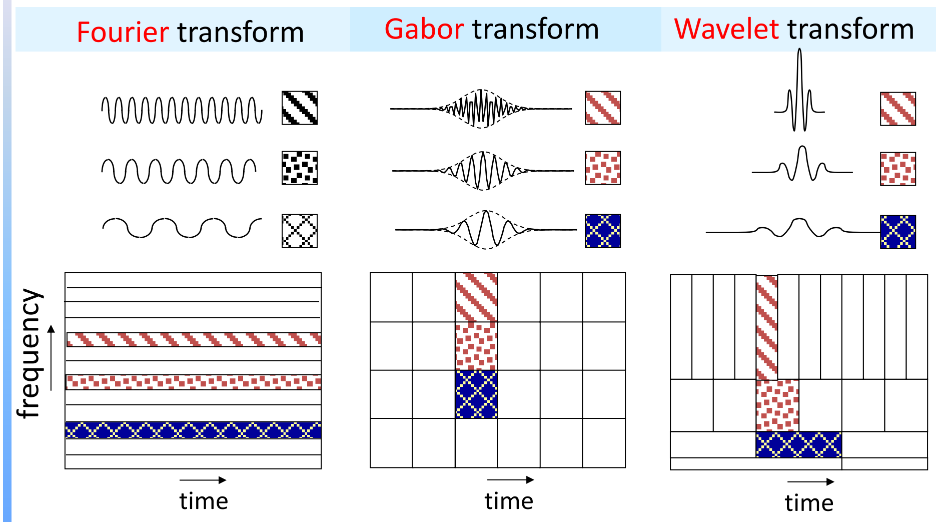
Why are wavelets useful?...

- In signal processing, the representation of signals plays a major role
- The Fourier representation reveals the spectral content of a signal but makes it impossible to reveal a particular moment in time (or space) where a certain change has occurred
  - ➢Fourier analysis is not suitable for transient signals
- In Short-Time Fourier transform (STFT) or Gabor transform:

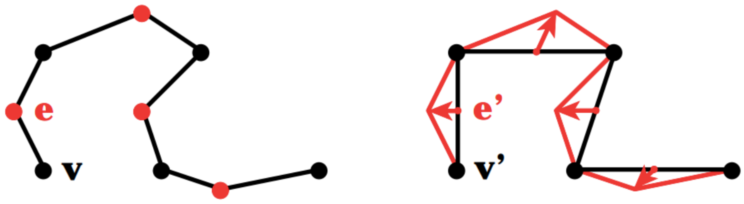
$$S(\tau,\omega)=\int_{-\infty}^{\infty} f(x)g(x-\tau)e^{-j\omega x}dx$$

the modulation frequency is changing but the window remains fixed.

...Why are wavelets useful?...



Subdivision using wavelets



Wavelet decomposition coarsens a control polygon and stores difference vectors in place of removed vertices.

This principle can be combined with different subdivision surfaces.

Second Generation Wavelets - Lifting scheme

Consider $\mathbf{x} = (x_k)_{k \in \mathbf{Z}}$ with $x_k \in \mathbf{R}$.

Split into two disjoint sets which are called **polyphase** components:

- The **even** indexed samples ("evens"): $\mathbf{x}_e = (x_{2k})_{k \in \mathbf{Z}}$
- The **odd** indexed samples ("odds"): $\mathbf{x}_o = (x_{2k+1})_{k \in \mathbf{Z}}$

Denote by $P(\mathbf{x}_e)$ predicted odds from the evens and record the difference:

$$\mathbf{d} = \mathbf{x}_o - P(\mathbf{x}_e)$$

Given the detail and the evens, we can recover the odds as

$$\mathbf{x}_o = P(\mathbf{x}_e) + \mathbf{d}$$

If P is a good predictor, d is sparse. An easy predictor of the odd sample x_{2k+1} is the average of the neighboring evens:

$$d_k = x_{2k+1} - (x_{2k} + x_{2k+2})/2.$$

Second Generation Wavelets - Lifting scheme

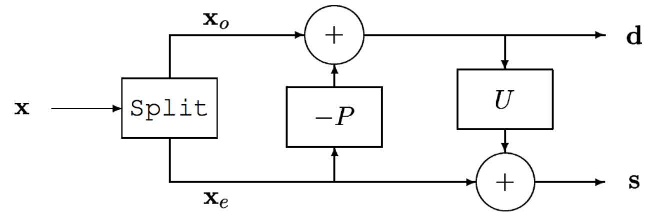
With the current scheme, the frequency separation between supposedly lowpass and bandpass parts is poor since $\mathbf{x}_e$ is obtained simply by subsampling.

To correct this, a second lifting is applied with an update operator U

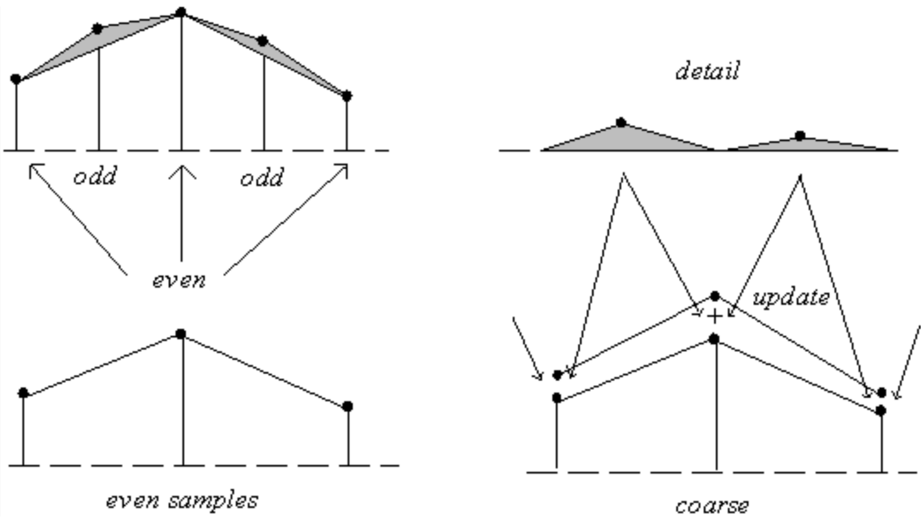
$$\mathbf{s} = \mathbf{x}_e + U(\mathbf{d})$$

We can recover $\mathbf{x}_e$ as

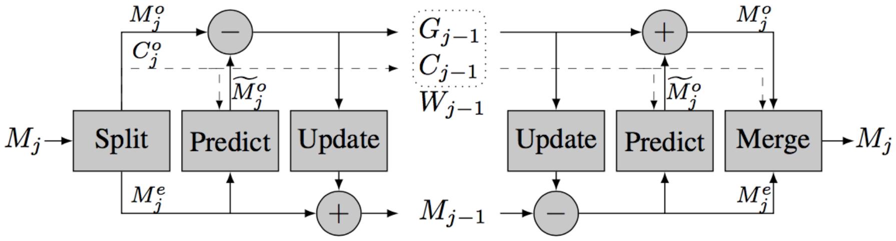
$$\mathbf{x}_e = \mathbf{s} - U(\mathbf{d})$$



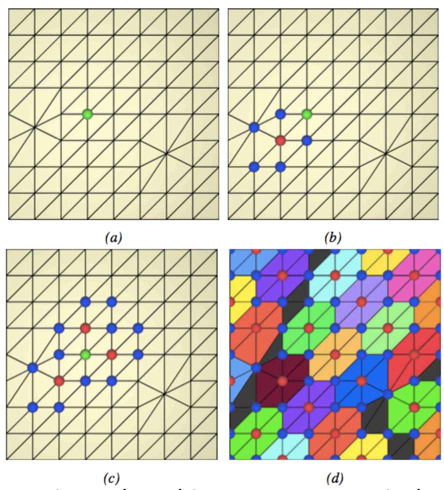
Second Generation Wavelets - Lifting scheme



Wavelet-based mesh coding

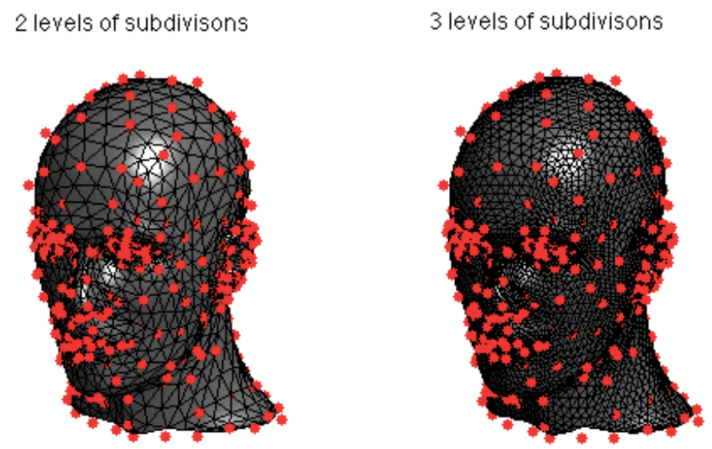


Wavelet-based mesh coding



Selecting odd and even vertices. The arbitrary start vertex is depicted in green, the odd and even vertices in red and blue, respectively. (d) shows the resulting patches. Image from the PhD thesis Jonas El Sayeh Khalil, Ghent University 2018.

Matlab Toolbox: Wavelets on Meshes

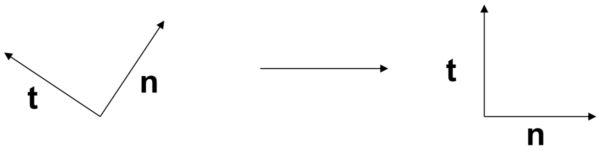


The Effect of a Transformation on a Normal

- What happens to a normal vector under a linear transformation?
  - Translation?
  - Rotation?
  - Scaling?
  - Other?

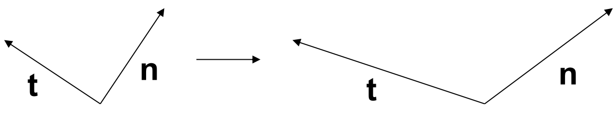
Rotation and Translation

- It seems intuitive that if **t** (tangent) and **n** (normal) are orthogonal, then their images will be orthogonal under both translation and rotation, since these are rigid motions.



Scaling

- However, under scaling their images might not be orthogonal.



Shear transform

- Nor will they be orthogonal under a shear transformation.
  - ➢ Re-compute normals!