Computer Graphics

Dr. rer. nat. Martin Möddel

April 6, 2021

Institut für Biomedizinische Bildgebung

Meshes in Computer Graphics

Meshes

Polygon Mesh
In computer graphics a polygon mesh is a collection of vertices, edges and faces (polygons) defining the surfaces of an object. Most often triangle meshes are considered

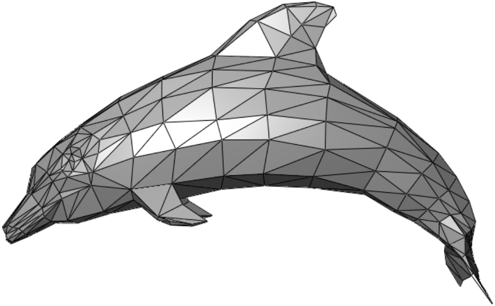


Figure 1: "An example of a polygon mesh" by en:User:Chrschn realeased into the public domain

Mesh Creation

- Mesh creation refers to the process of finding a suitable polygonal mesh for a given surface
- Common strategies are
    - Triangularization of parametric surfaces
    - Cutting cube triangularization of implicit surfaces
    - Marching triangularization of implicit surfaces
- Refining known meshes, such as the Platonic solids

Figure 2: "Platonic-solids set of five dice, (from left) tetrahedron (d4), cube (d6), octahedron (d8), dodecahedron (d12), and icosahedron (d20)." by unknown author licensed under https://creativecommons.org/licenses/by-sa/3.0/deed.en

Triangularization of parametric surfaces

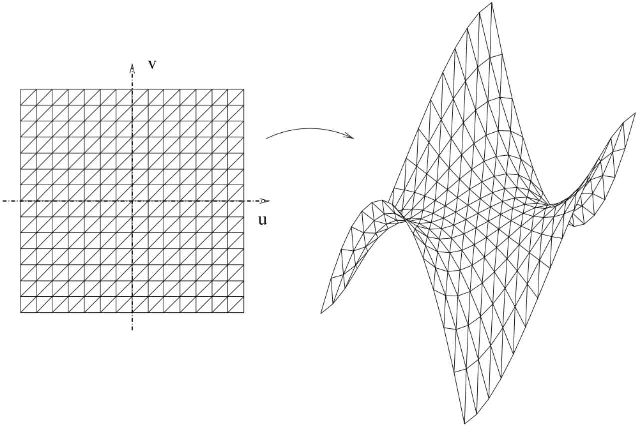


Figure 3: "Triangulierung einer parametrisierten Fläche (Affensattel)" by Ag2gaeh realeased under the CC BY-SA 4.0 license

Triangularization of parametric surfaces

- Triangularization of an explicitly defined parametric surface is quite straight forward
- At first a triangularization of the parameter space is required
- The vertices of the parameter space triangularization can then directly be mapped by the parametric mapping to define corresponding triangles in 3D space

Remark

- The mapping will usually change the size of the triangles
- Take care at parameter space boundaries, e.g. pole or Greenwich meridian arc for sphere

Cutting Cube Triangularization

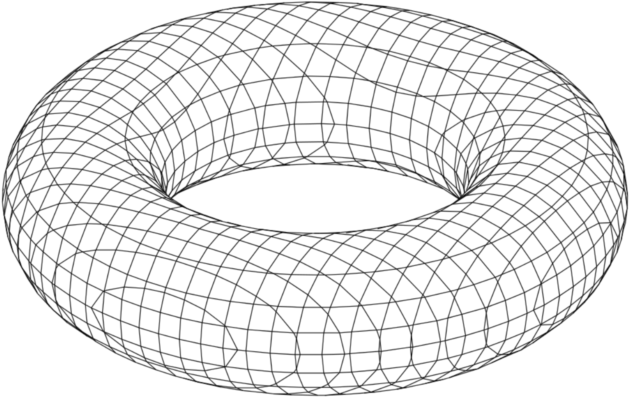


Figure 4: “Polygonisierung eines Torus mit der cutting cube Methode” by Ag2gaeh realeased under the CC BY-SA 4.0 license

Cutting Cube Triangularization

- Divide the 3D space into cubes
- Intersecting edges of the cubes with the implicit surface creates polygons on the surface
- The polygons can then further be subdivided into triangles

Remark

- Intersection problem might be difficult to solve
- Great overhead for managing the data

Marching Triangularization

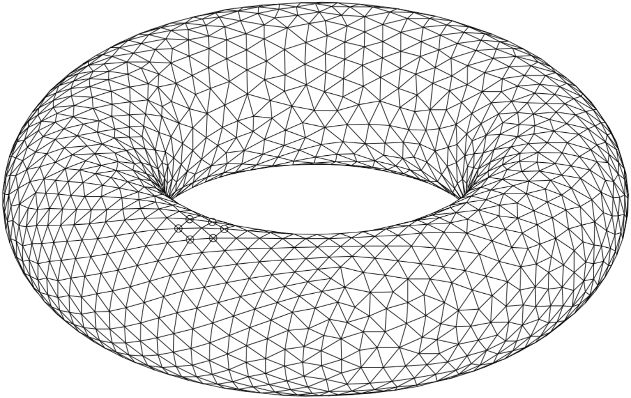


Figure 5: "Triangulation eines Torus" by Ag2gaeh realeased under the CC BY-SA 4.0 license

Marching Triangularization

Idea
A marching method for the triangulation of surfaces by Erich Hartmann (1998)

1. Choose a starting point on the surface and build a hexagon in tangent space and project it onto the surface
    - The triangles of the hexagon are the starting triangles
    - The six outer vertices create the first "primary outer front polygon" Π_0
2. Determine the angle of the area still to be triangulated at each vertex of Π_0
3. Check if any vertex of Π_0 is near a non-neighbouring point of Π_0 or a point of another front polygon Π_k, $k > 0$
    - In the first case divide the primary front polygon Π_0 into smaller parts
    - Else unite Π_0 and Π_k

4. Surround the point $p_m \in \Pi_0$ with minimal angle by isosceles triangles with approximately 60° angles and leg length δ_t, delete p_m from Π_0 and add the new points into Π_0
5. Repeat the steps 2-4 until there are only three points left in Π_0, which generate the final triangle
    - If there is still a front polygon left then take it as new front polygon Π_0 and proceed with steps 2-4
    - If there is no front polygons left then the triangularization is finished

Remark
For the algorithm to converge the surface should have finite size

Marching Triangularization

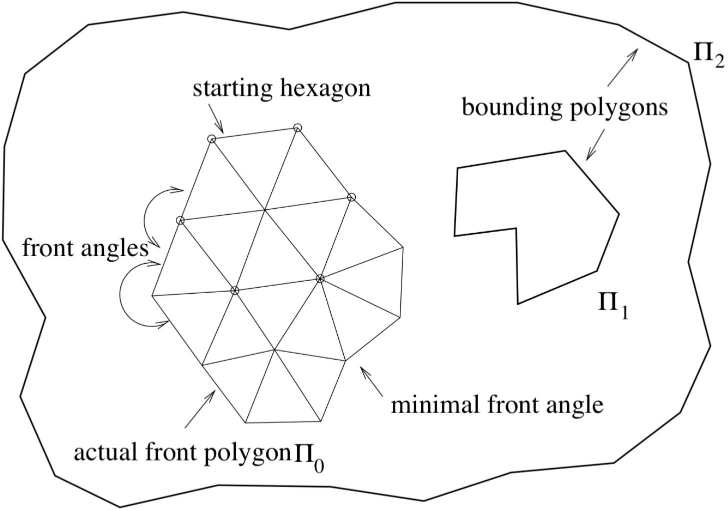


Figure 6: Pictures (in the following) taken fromwww.mathematik.tu-darmstadt.de/ ehartmann/cdgen0104.pdf

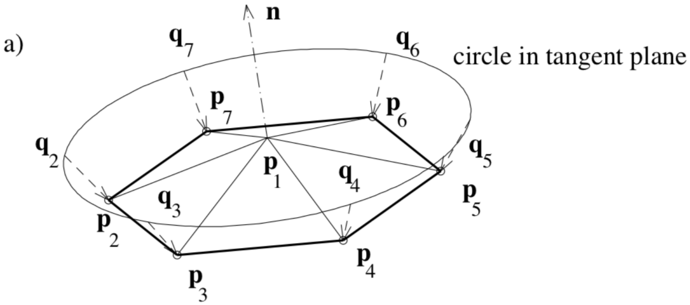
Marching Triangularization - Gradient Projection

- As the triangles are created in tangent space, we need a method to project the triangle vertices q_l from tangent space onto a surface points p_l
- Assume the surface is implicitly given by $f(x) = 0$ with non-zero gradient ∇f at any point q
- p, the corresponding surface normal and the two tangent vectors at p can be calculated by the following three steps

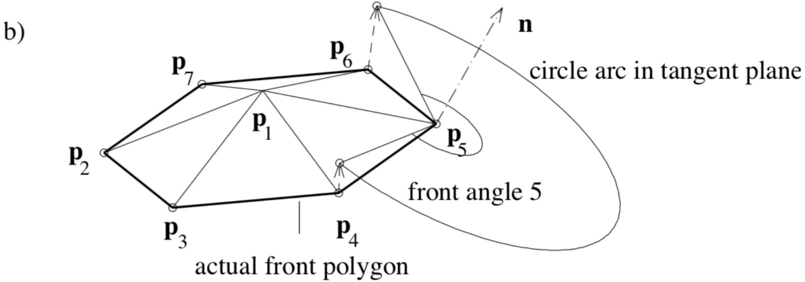
Marching Triangularization - Gradient Projection

1. Decent onto the surface along the gradient
   a $\mathrm{u}_0 = \mathrm{q}$
   b repeat $\mathrm{u}_{k+1} = \mathrm{u}_k - \frac{f(\mathrm{u}_k)}{\|\nabla f(\mathrm{u}_k)\|_2^2} \nabla f(\mathrm{u}_k)$ until $\|\mathrm{u}_{k+1} - \mathrm{u}_k\|_2 < \varepsilon$ for some preset $\varepsilon > 0$ and set $\mathrm{p} = \mathrm{u}_{k+1}$
2. The surface normal at p is $\mathrm{n} = (n_x, n_y, n_z) = \frac{\nabla f(\mathrm{p})}{\|\nabla f(\mathrm{p})\|_2}$
3. The tangent vectors are
   a $\mathrm{t}_1 = \frac{(n_y, -n_x, 0)}{\|(n_y, -n_x, 0)\|_2}$ if $n_x > 0.5$ or $n_y > 0.5$ else $\mathrm{t}_1 = \frac{(-n_z, 0, n_x)}{\|(-n_z, 0, n_x)\|_2}$
   b $\mathrm{t}_2 = \mathrm{n} \times \mathrm{t}_1$

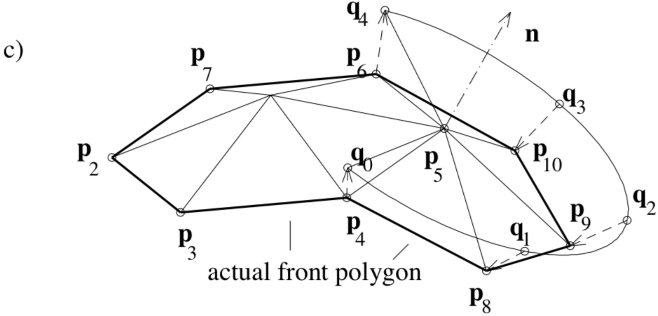
Marching Triangularization - Gradient Projection



Marching Triangularization - Gradient Projection



Marching Triangularization - Gradient Projection



Marching Triangularization - Hexagon Initialization

- Start at any point in the vicinity of the surface and determine the closest surface point p_0, the corresponding normal vector n_0 and tangential vectors t_{01} and t_{02}
- Calculate the vertices $\mathsf{q}_1, \ldots, \mathsf{q}_6$ of the initial hexagon in tangent space by

$$\mathsf{q}_i = \mathsf{p}_0 + \delta_t \cos((i-1)\pi/3)\mathsf{t}_{01} + \delta_t \sin((i-1)\pi/3)\mathsf{t}_{02} \quad (1)$$

- Project $\mathsf{q}_1, \ldots, \mathsf{q}_6$ onto the surface using the gradient projection to obtain the surface points $\mathsf{p}_1, \ldots, \mathsf{p}_6$
- Each point p_i has normal vector n_i and tangential vectors t_{i1} and t_{i2} assigned
- The initial six triangles have vertices $(\mathsf{p}_0, \mathsf{p}_1, \mathsf{p}_2)$, $(\mathsf{p}_0, \mathsf{p}_2, \mathsf{p}_3)$, $(\mathsf{p}_0, \mathsf{p}_3, \mathsf{p}_4)$, $(\mathsf{p}_0, \mathsf{p}_4, \mathsf{p}_5)$, $(\mathsf{p}_0, \mathsf{p}_5, \mathsf{p}_6)$, and $(\mathsf{p}_0, \mathsf{p}_6, \mathsf{p}_1)$
- The outer vertices define the primary front polygon $\Pi_0 = \{\mathsf{p}_1, \ldots, \mathsf{p}_6\}$

Marching Triangularization - Angle Determination

For each point $\mathsf{p}_i \in \Pi_0 = \{\mathsf{p}_1, \ldots, \mathsf{p}_N\}$ new or neighbouring a new point recalculate the front angle ω_i

- Set $\mathsf{v}_1 = \mathsf{p}_{i-1}$ if $i > 1$ or $\mathsf{v}_1 = \mathsf{p}_N$ if $i = 1$
- Set $\mathsf{v}_2 = \mathsf{p}_{i+1}$ if $i < N$ or $\mathsf{v}_2 = \mathsf{p}_1$ if $i = N$
- Express both vertices in the local coordinate system of the i-th vertex

$$\mathsf{v}_1 = \mathsf{p}_i + \eta_1 \mathsf{n}_i + \tau_1 \mathsf{t}_{i1} + \vartheta_1 \mathsf{t}_{i2} \tag{2}$$

$$\mathsf{v}_2 = \mathsf{p}_i + \eta_2 \mathsf{n}_i + \tau_2 \mathsf{t}_{i1} + \vartheta_2 \mathsf{t}_{i2} \tag{3}$$

- Calculate the polar angle of the two vertices in tangent space $\varphi_1 = \mathrm{atan2}(\tau_1, \vartheta_1)$ and $\varphi_2 = \mathrm{atan2}(\tau_2, \vartheta_2)$
- The front angle is then given by

$$\omega_i = \begin{cases} \varphi_2 - \varphi_1 & \varphi_2 \geq \varphi_1 \\ 2\pi + \varphi_2 - \varphi_1 & \text{else} \end{cases} \tag{4}$$

Marching Triangularization - Avoid Overlap

Splitting

- Check if there are any $\mathsf{p}_i, \mathsf{p}_j \in \Pi_0$, $i < j$ such that $\|\mathsf{p}_i - \mathsf{p}_j\| < \delta_t$ and the vertices are neither next nor nearest next neighbours
- If so split Π_0 into the new $\Pi_0 = \{\mathsf{p}_1, \ldots, \mathsf{p}_i, \mathsf{p}_j, \ldots, \mathsf{p}_N\}$ and a new front polygon $\Pi_{\text{new}} = \{\mathsf{p}_i, \ldots, \mathsf{p}_j\}$
- Exclude p_i and p_j from **any** later (not only the present iteration) distance checks
- Repeat this check until no more pair is found
- Recalculate the angles at p_i and p_j in Π_0

Marching Triangularization - Avoid Overlap

Joining

- Check the distance of Π_0 to all further front polygons Π_k, $k > 0$
- If there are $\mathrm{p}_i \in \Pi_0$ and $\mathrm{r}_j \in \Pi_k = \{\mathrm{r}_1, \ldots, \mathrm{r}_M\}$ with $\|\mathrm{p}_i - \mathrm{r}_j\|_2 < \delta_t$ then the union of Π_0 and Π_k defines the new primary front polygon

$$\Pi_0 = \{\mathrm{p}_1, \ldots, \mathrm{p}_i, \mathrm{r}_j, \ldots, \mathrm{r}_M, \mathrm{r}_1, \ldots, \mathrm{r}_j, \mathrm{p}_i, \ldots \mathrm{p}_N\} \tag{5}$$

- Calculate the front angles at p_i and r_j at their first appearance
- Surround the vertex with smallest angle with triangles first
- Surround the remaining vertex next

Marching Triangularization

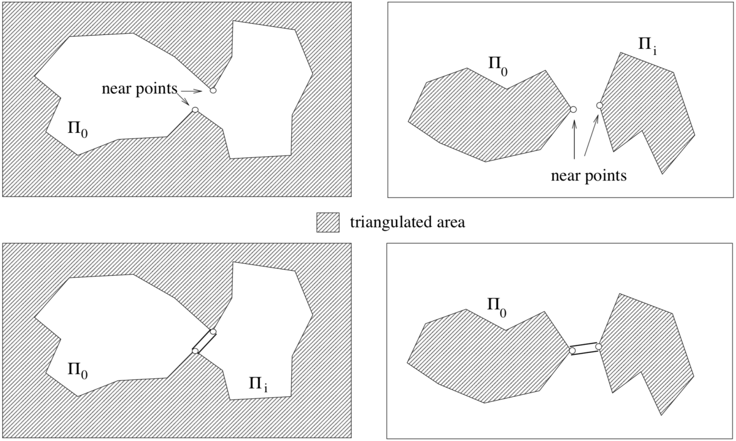


Figure 7: Dividing (left) and uniting (right) the actual front polygon.

Marching Triangularization - Avoid Overlap

Remarks

- Sometimes the line segment between p_i and r_j intersects an already triangulated area (this must be detected and avoided)
  - To determine this case we can compare the front angle ω_i at vertex p_i with $\tilde{\omega}_i$, which is calculated using $v_2 = r_j$ for the angle determination
  - p_i and r_j are connected through an already triangulated area, if $\omega_i < \tilde{\omega}_i$
  - No joining in this case
- Add and remove vertices during the triangle surrounding step as discussed below

Marching Triangularization - Triangle Creation

1. Consider $\mathrm{p}_i \in \Pi_0$ with minimal front angle ω_i
2. Determine its neighbours v_1 and v_2 as discussed above
3. Determine the number of triangles to be created

$$n_t = \left\lfloor \frac{3\omega_i}{\pi} \right\rfloor + 1 \text{ and } \Delta\omega = \omega_i / n_t \tag{6}$$

a If $\Delta\omega < 0.8$ and $n_t > 1$ then $n_t \to n_t - 1$ and $\Delta\omega = \omega_i / n_t$

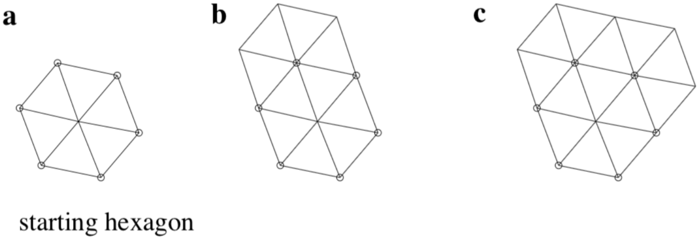
b If $n_t = 1$ and $\Delta\omega > 0.8$ and $\|\mathrm{v}_1 - \mathrm{v}_2\|_2 > 1.25\delta_t$ then $n_t = 2$ and $\Delta\omega \to \frac{\Delta\omega}{2}$

c If $\Delta\omega < 3$ and either $\|\mathrm{v}_1 - \mathrm{p}_i\|_2 \leq 0.5\delta_t$ or $\|\mathrm{v}_2 - \mathrm{p}_i\| \leq 0.5\delta_t$ then $n_t = 1$ and $\Delta\omega = \omega_i$

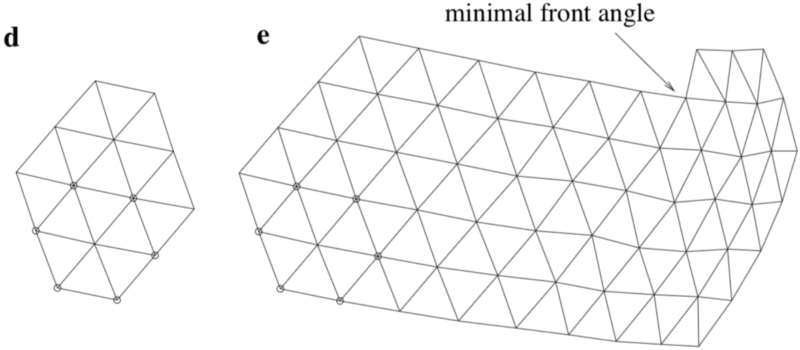
Marching Triangularization - Triangle Creation

4. If $n_t = 1$ generate the triangle with vertices v_1, v_2 and p_i, else
   a Project v_1 and v_2 onto the tangent plane at p_i, i.e. onto q_0 and q_{n_t} respectively, where
   $$\mathrm{q}_j = \mathrm{p}_i + \tau_j \mathrm{t}_{i1} + \vartheta_j \mathrm{t}_{i2} \tag{7}$$
   b The remaining q_k, $k = 1, \dots, n_t - 1$ are calculated by rotating $\mathrm{p}_i + \delta_t \frac{(\mathrm{q}_0 - \mathrm{p}_i)}{\|\mathrm{q}_0 - \mathrm{p}_i\|_2}$ by the angle $k\Delta\omega$ around the surface normal at p_i
   c Gradient projection of q_k onto the surface yields p_{N+k} for all $k = 1, \dots, n_t - 1$
   d The n_t new triangles have vertices $(\mathrm{p}_i, \mathrm{v}_1, \mathrm{p}_{N+1}), (\mathrm{p}_i, \mathrm{p}_{N+1}, \mathrm{p}_{N+2}), \dots, (\mathrm{p}_i, \mathrm{p}_{N+n_t-1}, \mathrm{v}_2)$
5. Delete p_i from Π_0 and insert the points $\mathrm{p}_{N+1}, \dots, \mathrm{p}_{N+n_t-1}$ at its position (no points added for $n_t = 1$)

Marching Triangularization - Example Sphere



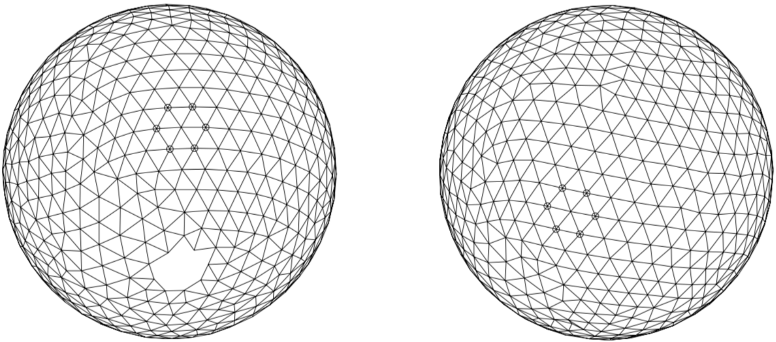
Marching Triangularization - Example Sphere



Marching Triangularization - Example Sphere

f

g



still a hole

final triangulation

Remarks

- Apart from visualization mesh creation plays an important role in the application of finite element methods
- The methods presented are conceptionally easy to understand
- There are much more elaborate methods available, e.g. those which adapt the size of the triangles to the local curvature of the surface

Triangle Meshes

Triangle Meshes

- As many models in computer graphic are composed of triangular meshes efficient handling of these meshes is crucial
- Apart from transferring meshes in between graphics application and graphics pipeline one might want to draw, subdivide and edit the meshes
- For some applications not only the triangles are required, but also adjacency information, i.e. information about shared vertices and edges or neighbouring triangles
- Apart from the minimal information (triangles and vertices) one often stores additional information assigned to the vertices or triangle faces

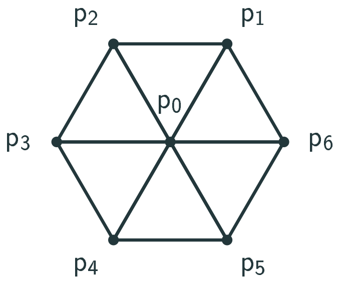
Triangle Meshes - Topology

- In mesh topology one is concerned with the properties of the mesh that are preserved under continuous deformations
- A common assumption is that meshes are manifolds
  - Each edge is shared by exactly two triangles
  - Each vertex has a single complete loop of triangles around it
- A relaxation of this topology is a manifold with boundary where
  - Each edge is used by either one or two triangles
  - Each vertex connects to a single edge-connected set of triangles
- Another topological property the orientation of the mesh which allows to distinguish front and back side
  - The front of a triangle is, where the vertices are arranged in counter clockwise order (right hand rule)
  - A mesh is consistently oriented if all pair of adjacent triangles within the mesh agree on front side

Triangle Meshes - Indexed Mesh Storage

- The simplest storage form of a triangle mesh is to store the three vertex positions for each triangle
- However since most vertices are used multiple times by multiple triangles we can do better by using an indexed mesh
  - Store the vertices in a list
  - Store three indices per triangle
- Assuming n_t triangles, n_v vertices and the same storage requirements for floats and indices we need
  - Three vertices per triangle ($9n_t$ in total)
  - One vector per vertex and three indices per triangle ($3n_v + 3n_t$ in total)
  - For large meshes $n_t \approx 2n_v$ (each vertex connected to six triangles) the storage requirements are reduced by about two

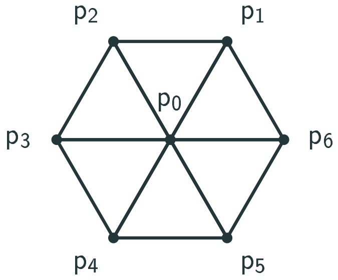
Triangle Meshes - Indexed Mesh Storage



Separate Triangles

#t	Vertices
0	(p_0, p_1, p_2)
1	(p_0, p_2, p_3)
2	(p_0, p_3, p_4)
3	(p_0, p_4, p_5)
4	(p_0, p_5, p_6)
5	(p_0, p_6, p_1)

Triangle Meshes - Indexed Mesh Storage



Shared Vertices

#v	0	1	2	3	4	5	6
Pos	p_0	p_1	p_2	p_3	p_4	p_5	p_6

#t	Vertices
0	$(0, 1, 2)$
1	$(0, 2, 3)$
2	$(0, 3, 4)$
3	$(0, 4, 5)$
4	$(0, 5, 6)$
5	$(0, 6, 1)$

Triangle Meshes - Triangle Strips and Fans

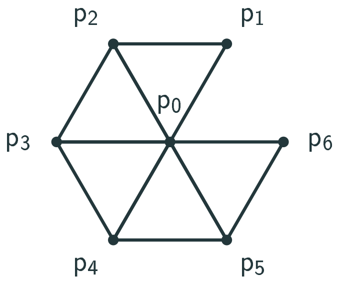
If we want an even more compact representation of our mesh we can use triangle strips and triangle fans

Triangle Fan

- In a triangle fan all triangles share a common vertex
- The other vertices generate triangles like the vanes of a fan
- A fan can be described by an ordered list of indices

Triangle Stripe

- In a triangle stripe we start with a single triangle
- New vertices are added alternating top and bottom to define the next triangle
- Here too an ordered list of indices suffices



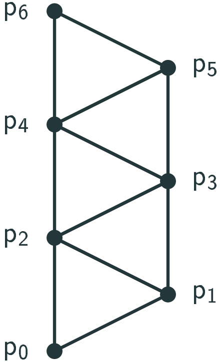
Triangle Fans

#v	0	1	2	3	4	5	6
Pos	p_0	p_1	p_2	p_3	p_4	p_5	p_6

The sequence $(0, 1, 2, 3, 4, 5, 6)$ specifies the triangles

#t	Vertices
0	$(0, 1, 2)$
1	$(0, 2, 3)$
2	$(0, 3, 4)$
3	$(0, 4, 5)$
4	$(0, 5, 6)$

Triangle Meshes - Triangle Strips and Fans



Triangle Strips

#v	0	1	2	3	4	5	6
Pos	p_0	p_1	p_2	p_3	p_4	p_5	p_6

The sequence $(0, 1, 2, 3, 4, 5, 6)$ specifies the triangles

#t	Vertices
0	$(0, 1, 2)$
1	$(1, 3, 2)$
2	$(2, 3, 4)$
3	$(3, 5, 4)$
4	$(4, 5, 6)$

Triangle Meshes - Mesh Connectivity

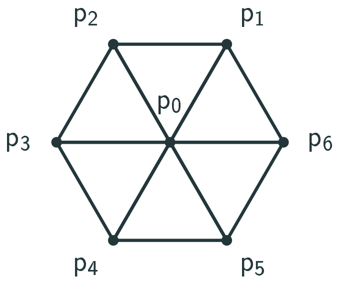
- To modify or edit meshes it is often necessary to obtain connectivity information
  - Which triangles are adjacent to a given one
  - Which triangles share a specific edge
  - Which triangles share a certain vertex
  - Which edges share a given vertex
- Obtaining this information from the data structures discussed so far is computationally demanding
- Storage of all these informations seem to costly in terms of memory
- There are data structures which allow to obtain connectivity information
  - Triangle-neighbour structure
  - Winged-edge structure

Triangle Meshes - Triangle-Neighbour Structure

- By modifying the indexed mesh storage structure we can ensure that all of the queries are answered in constant time
- Add a pointer to one adjacent triangle for each vertex
- Add pointers to all three neighbouring triangles for each triangle
- Store the pointers in such an order, that the k-th pointer points to the neighbouring triangle, which shares vertices k and $k+1$ with the current triangle
- This structure allows to efficiently answer the questions above by clever movement along the mesh

Triangle Meshes - Triangle-Neighbour Structure

Part of a Large Triangle Mesh



Triangle neighbour structure

#v	0	1	2	3	4	5	6
Position	p_0	p_1	p_2	p_3	p_4	p_5	p_6
ΔPointer	0	·	·	·	·	·	·

#t	Vertices	Neighbours
0	$(0, 1, 2)$	$(5, \cdot, 1)$
1	$(0, 2, 3)$	$(0, \cdot, 2)$
2	$(0, 3, 4)$	$(1, \cdot, 3)$
3	$(0, 4, 5)$	$(2, \cdot, 4)$
4	$(0, 5, 6)$	$(3, \cdot, 5)$
5	$(0, 6, 1)$	$(4, \cdot, 6)$

Adjacency Query

Input: vertex index i

function FindAdjacentTriangles(i)
- $t_i = \Delta\text{Pointer}[i]$
- $t_0 = t_i$
- **repeat**
  - Add t_0 to output list
  - Find i in t_0's vertices
  - Assign the corresponding vertices-index to j
  - Assign the j-th neighbouring triangle of $t_0 \to t_0$
- **until** t not t_i

end function