2. dafaq is polygonal modeling anyway?

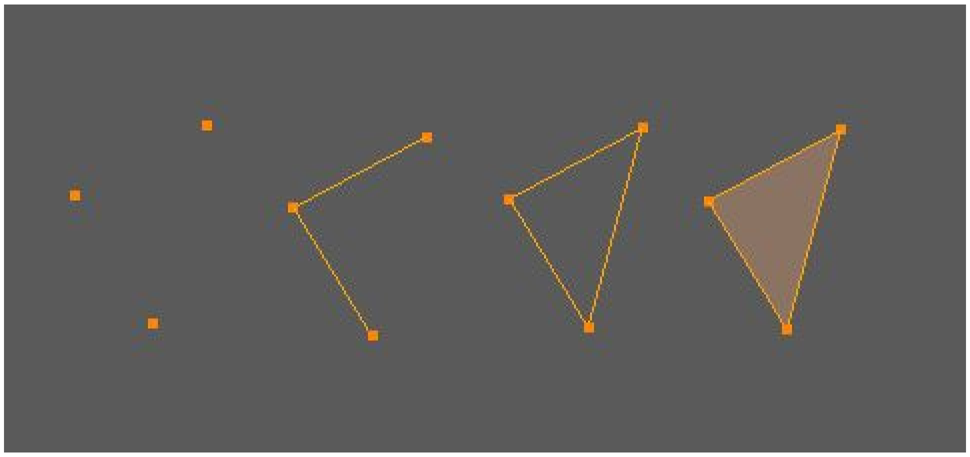
You need to somehow represent a three-dimensional object as data that can be stored inside a computah, manipulated as needed, and projected onto 2d plane (e.g. screen or image file). There's just one wee lil problem. Physical objects in real reality are kind of continuous, as far as human aye is concernerded. And computahs can store only this much discrete values, let alone process in any reasonable amount of time.

So one way to approach the task is to approximately represent object's actual surface using connected portions of planes, i.e. polygons. Ranging from just a few dozen polygons, to uncountable fuckloads of them. In any given case, the actual number of polygons depends on desired level of detail, and is limited by available computational power, and the time humans are willing to spend doing their job.

The most basic element of a 3D polygonal mesh is a **vertex** - a point in space, defined by three numbers - its **x**, **y** and **z** coordinates. Except for those 3 numbers, a vertex can't have other properties - it doesn't have size, it doesn't have orientation. A vertex is an abstract concept. Yeah, refreshing elementary school math here, forgive me for assuming you didn't know that, please don't take this personally.
Line segment that connects two vertices is an **edge**. Again, just as a line, edge is an abstract concept. Despite our 3d software highlights edges, so it is possible to work with them (obvjastly), edges don't have thickness, so they can't actually be seen.
And portion of a plane bounded by three or more edges is a **face**. The face is the most basic element that does can be seen. Face also has no thickness, and has only one side (as in side of the moon, not side of a triangle). For any face, **a normal** is calculated - a vector that is perpendicular to a given face, and points in either one of two possible directions. When looking from the direction to where **the normal** points, it is face's business side, and its other side you should never show to

anybody (even your family doctor, ESPECIALLY your family doctor), because technically it kind of doesn’t exist. I guess.

And a polygon is n vertices + n edges + one face, as a whole.



// on the very left - three vertices not connected by edges
// on the very right - three vertices, connected by edges and filled with a face, together comprising a polygon

So the job of a modeler is manipulating vertices, edges and faces, to represent an object as accurately as needed, using as little polygons as possible, while keeping the mesh well-structured. Structure of a polygonal mesh is referred to as **topology**. (and when topology is good, it is referred to as good).

So what defines good topology?
First, there are rules that are based on the way the computah processes mesh data, which in turn is based on laws of geometry and mathematics.
Second, there are rules based on intended use for the mesh (for example, it should deform correctly during animation).
Third (and this is related to both the first and the second), depending on the arrangement of polygons, a mesh may be easier or harder to work with, i.e. with proper topology it may be still relatively easy to apply major changes to complex models.

So basically, bad topology can either produce unexpected weird-looking results on various stages (rendering, animation, further modeling), or make your mesh unmanageable, or both. Obviously, good topology allows you to avoid all that.

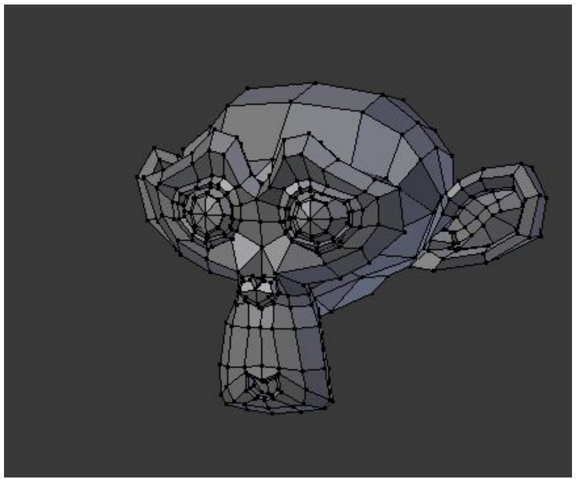
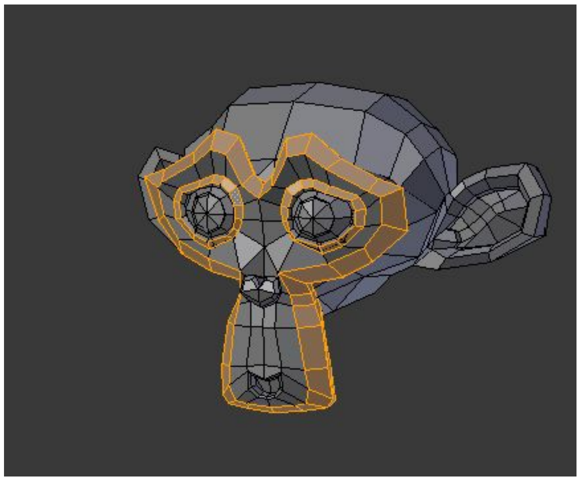
So here are some guidelines. Please note, that they are not detailed, and not even particularly accurate. But trust me, that’s more than you want to know right now, and going into details would stretch this paper too long. I know that you don’t have any friends, and there’s nothing better for you to do, but that would be just too irrelevant for you now. You will gradually pick up many nuances of this fackering later as you progress, probably without much effort.

Okay, so.
No less than three non-colinear vertices can unambiguously define a plane (and, therefore, form a polygon). **No more** than three vertices are guaranteed to always be co-planar. So triangles. Triangles. I guess that’s why.

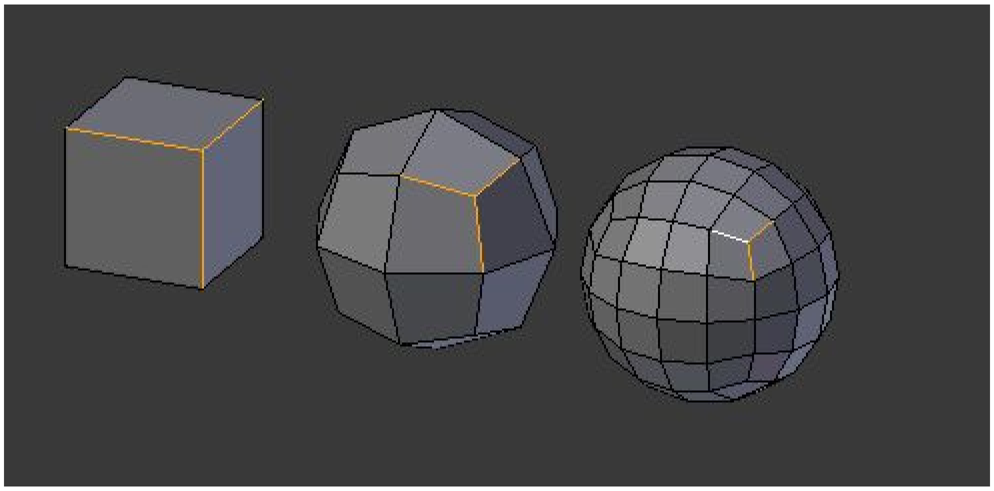
A polygon with four sides is a **quad**, a triangle is a **triangle**, and if the number of sides is anything greater than 4, it is referred to as an **N-gon**.

Computer hardware works only with triangles, humans mostly want to work with quads, and no one wants to be friends with you, or work with N-gons.

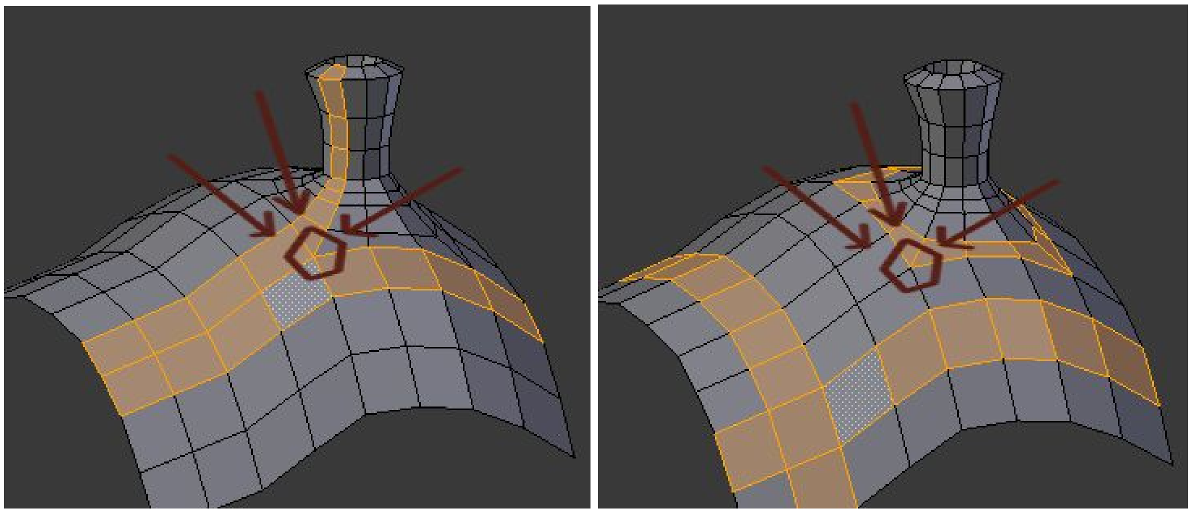
So you arrange quads in such a way, that continuous loops of edges/faces kind of repeat the form of the object, as shown on this Blender mascot monkey (her name is Suzanne, and I haven’t just made this up)

While working with quads, you would want any one vertex to be shared by 4 faces (i.e. have 4 edges converging to it), for the most of your mesh. But in certain places you will need to have vertices that share either 3 or 5 faces.

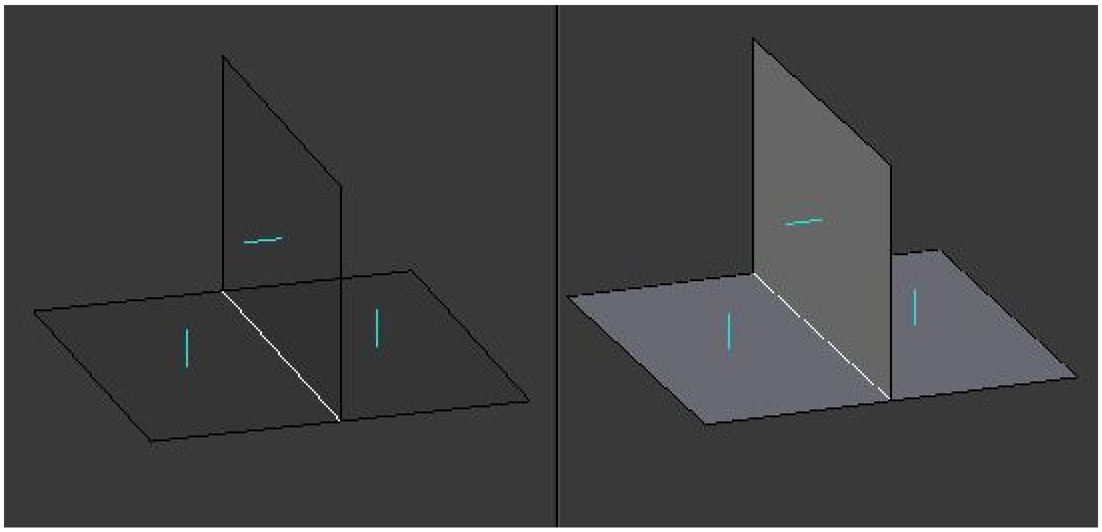


like on this subdivided cube



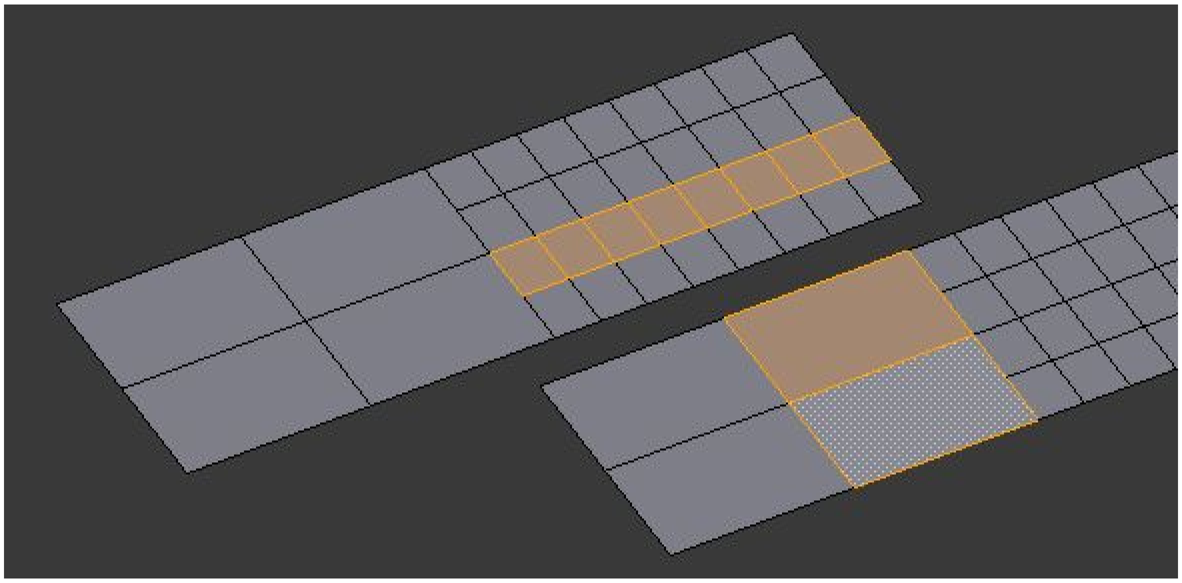
or on this random curved surface thing that has random cylindrical thing poking out of it

More than 5 edges converging to one vertex, unless situated on completely planar surface, is often a bad idea, and should be used carefully.

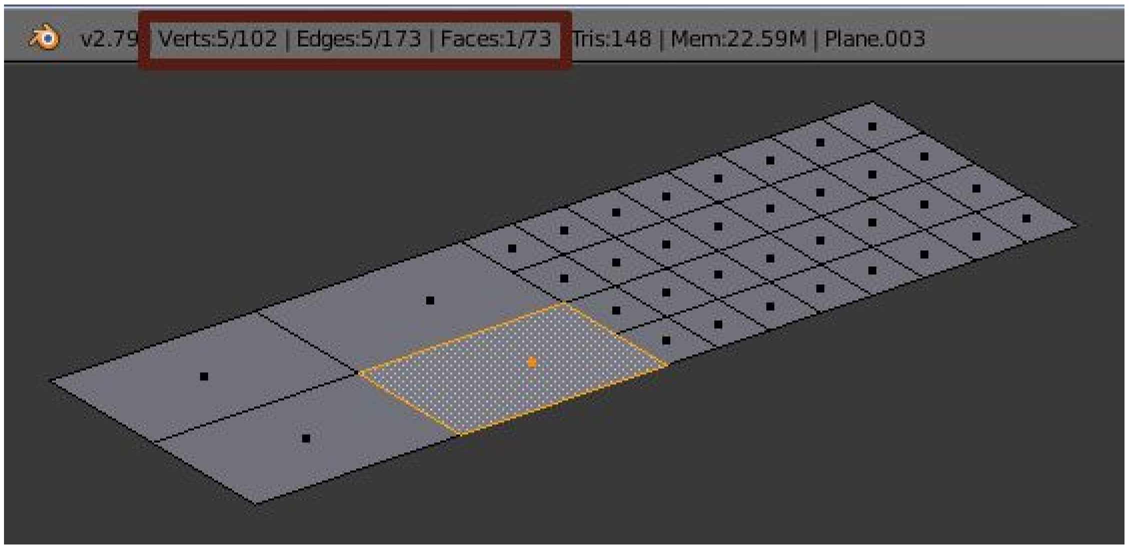


It is completely illegal to have THIS. Completely as in absolutely, for sure, hugely, enormously illegal. Of course, Blender might let you do that, and some exceptionally bright individuals may find an actual ultra-unorthodox use for it. But it just doesn’t make any sense in terms of anything. Edge should never share three or more faces.

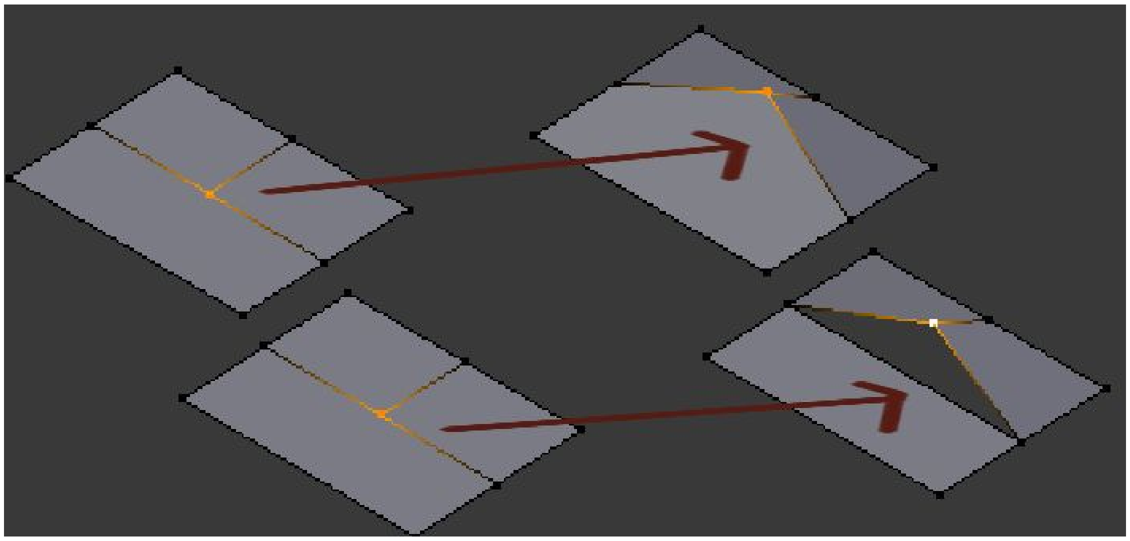
And one more thing.

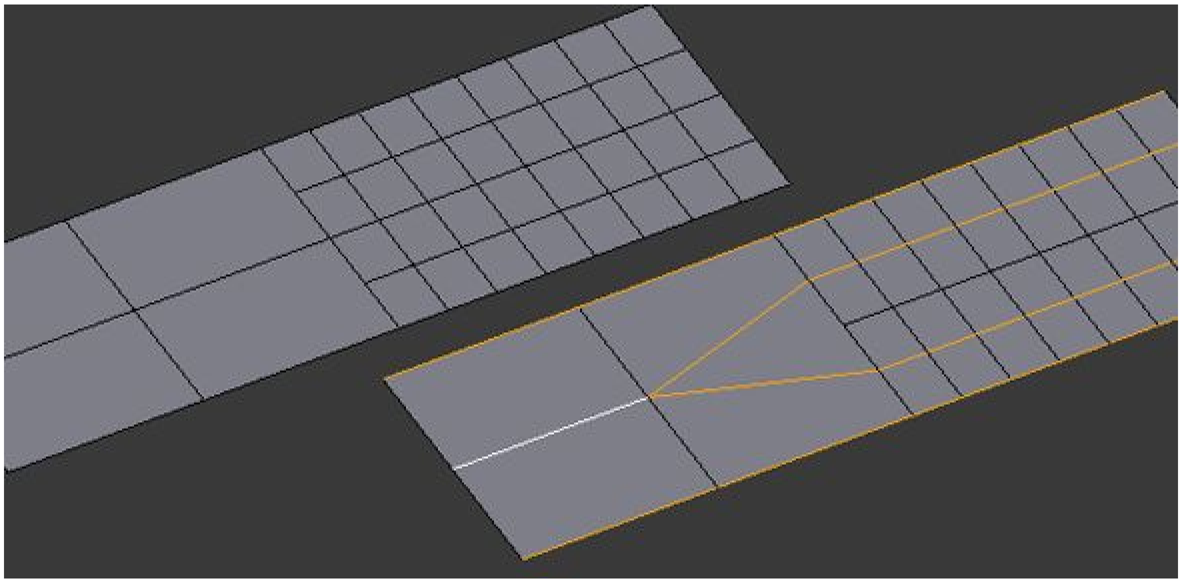


You almost never want to have this. This can be a result of one of the following:


v2.79 | Verts:5/102 | Edges:5/173 | Faces:1/73 | Tris:148 | Mem:22.59M | Plane.003

1) either these bigger rectangles are actually 5-sided n-gons,
2) or these vertices in the middle are separated from the rest of the mesh, and you have three adjacent edges located along one line, and overlapping each other.
These two look similar in edit mode, but may produce different “glitches” down the road.


//upper is illustration for 1), lower is for 2)


And this is one of the ways of doing an all-quad transition between different mesh densities.

I guess that’s it for now.
What you have just read is not specific to Blender. These are fundamentals of polygonal modeling in general, and thus they are applicable to any other similar software.