Manual Virtual Character Modelling

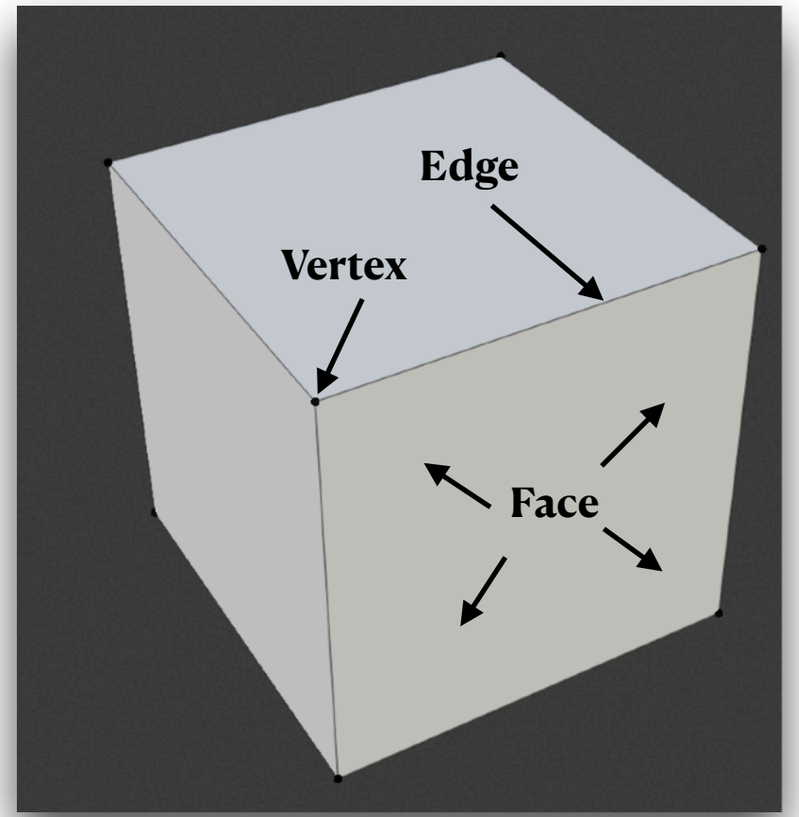
IAPV

INSTITUTO UNIVERSITÁRIO DE LISBOA

Meshes

Structure - Basics

- A **vertex** represents a point in a 3D space
- An **edge** connects two vertices by a straight line
- A **face** is the area between three (triangle), four (quad), and n>4 (*n-gon*) vertices
- To a set of connected vertices, edges, and faces we call **mesh**
- Quads and *n-gons* are usually triangulated for real-time rendering purposes (by GPU)
- That is because triangles are fast to compute given that they are always **flat**

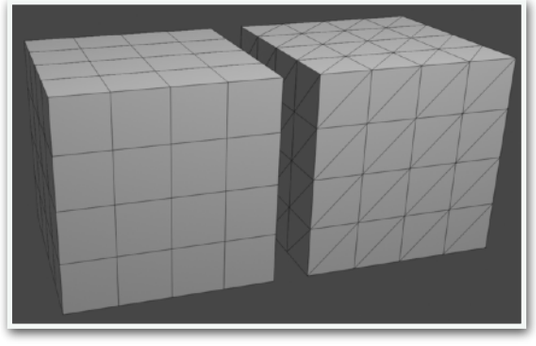


Meshes

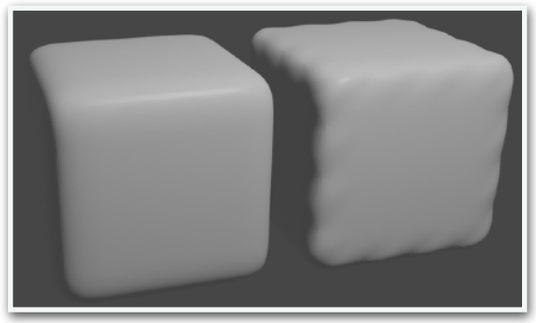
Structure - Basics

- A **vertex** represents a point in a 3D space
- An **edge** connects two vertices by a straight line
- A **face** is the area between three (triangle), four (quad), and n>4 (*n-gon*) vertices
- To a set of connected vertices, edges, and faces we call mesh
- Quads and *n-gons* are usually triangulated for real-time rendering purposes (by GPU)
- That is because triangles are fast to compute given that they are always **flat**
- However, for animation/modelling purposes, **quads are often preferred** over triangles and *n-gons*
- Deforming, smoothing, and loop selection operations **work better with quad-based meshes**

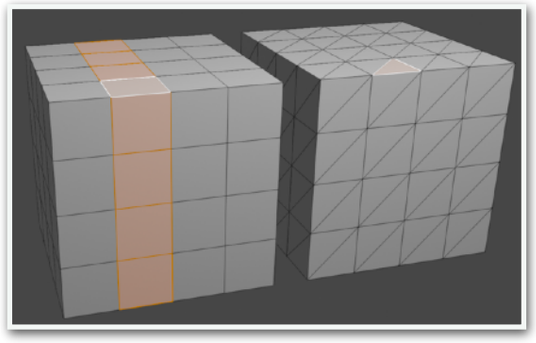
Original Meshes



Smoothed Meshes



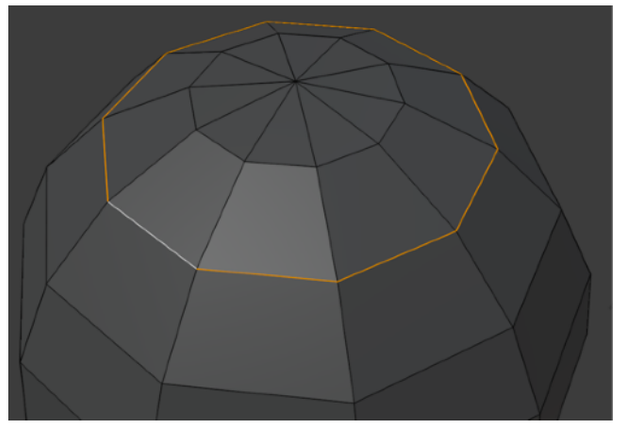
Loop face selection



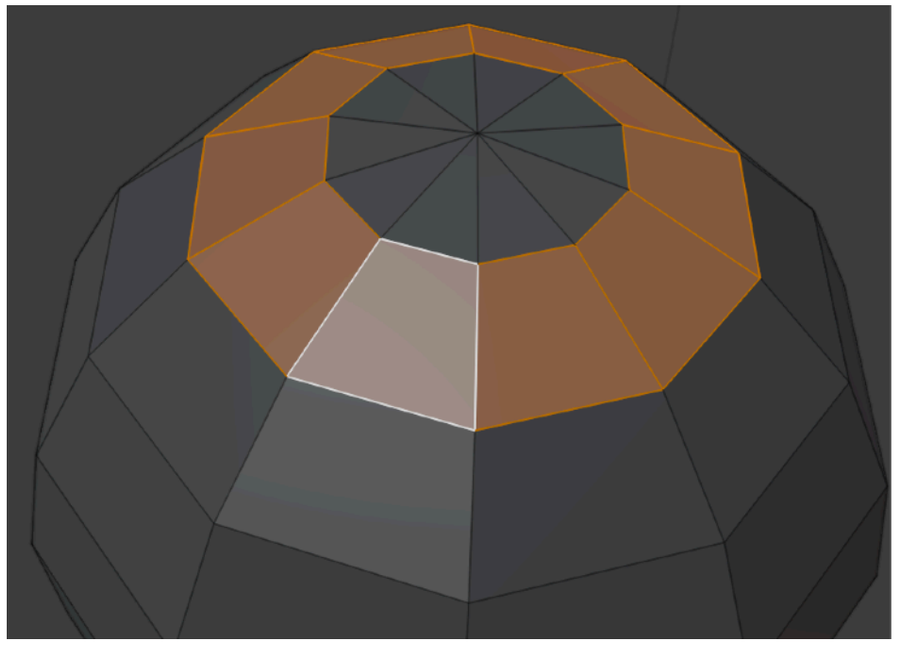
Meshes

Structure - Loops

- An **edge loop** is a path of **connected edges**, in which the path follows the edge “in the middle” every time a **four-way junction** is met during the path construction process
- The path construction process **stops whenever it finds a pole** (a vertex connected to three, five or more edges)
- If the initial edge is a boundary edge (connected to a single face), then the path follows it along the connected **boundary edge**
- **Face loops** can also be defined as the set of **faces between two edge loops**

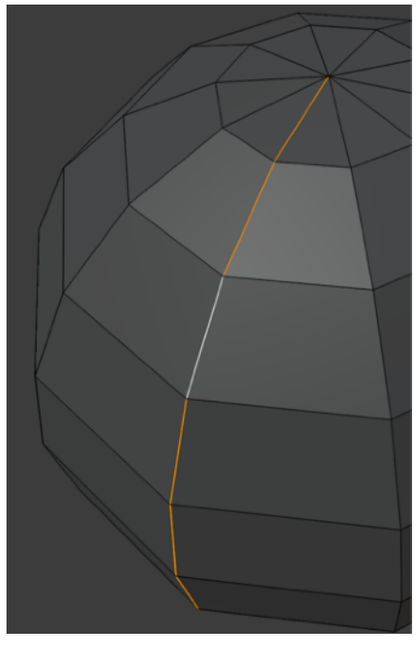


A cyclic edge loop
(Initial edge in white)



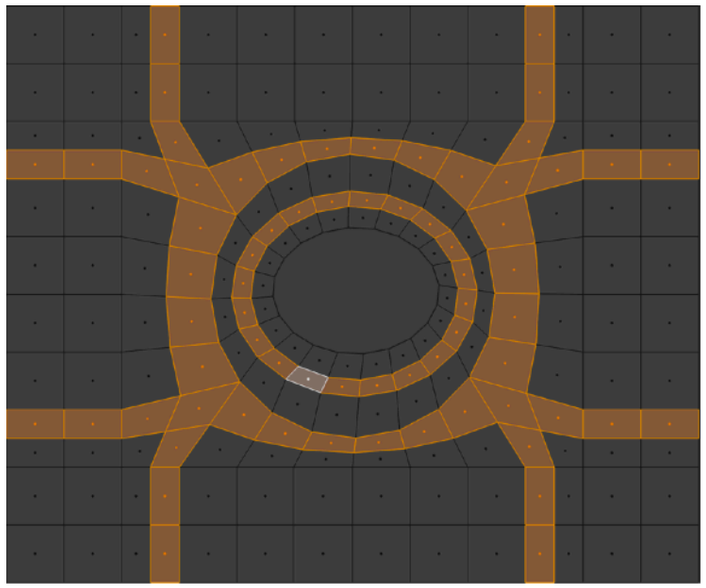
A cyclic face loop
(Initial face in white)

An acyclic edge loop due to the presence of poles
(Initial edge in white)

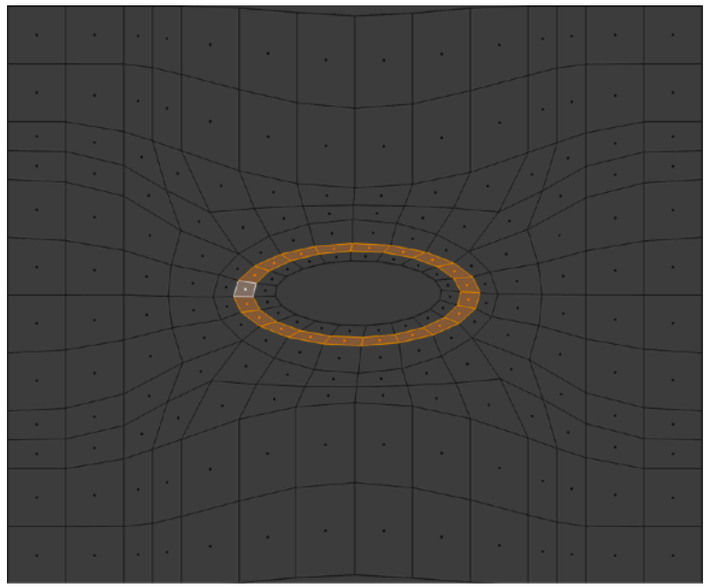


Meshes

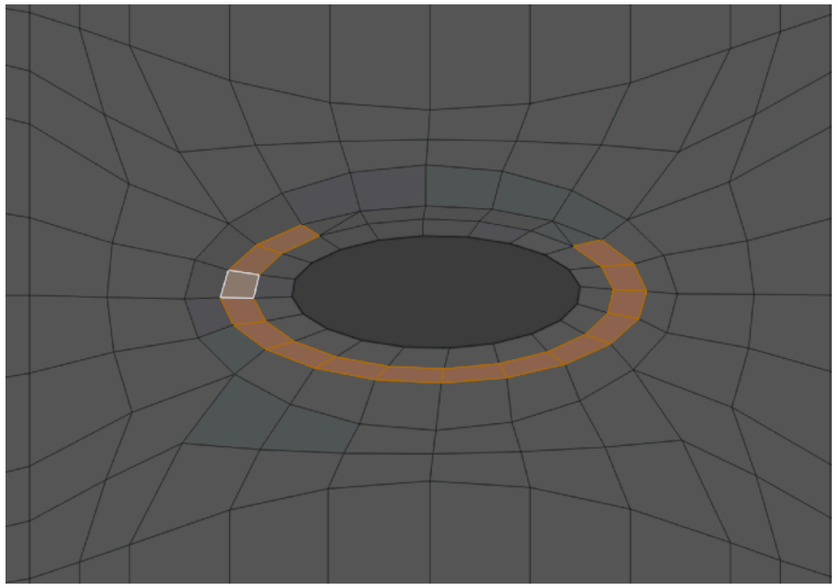
Structure - Loops - example



A mesh with an **organic appearance** (e.g., an eye) with multiple **face loops** simultaneous selected. Note how these loops naturally follow the organic structure of the mesh.



A single face loop selected and then scaled along the vertical axis around the face's center of mass. Note how naturally the scaling is performed, resembling the effect of a **muscle contraction**.

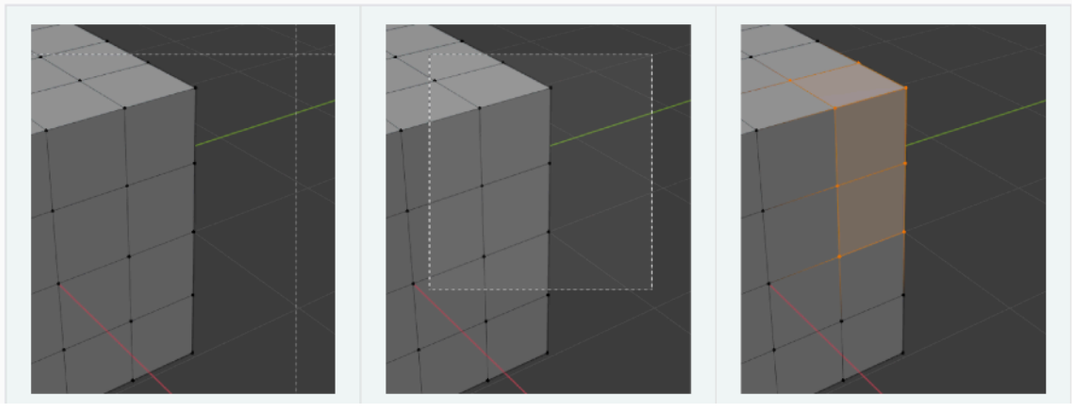


The same mesh with two faces triangulated, which **breaks** the ability to close the face loop. This shows how important is to **ensure a proper topology** of the designed models.

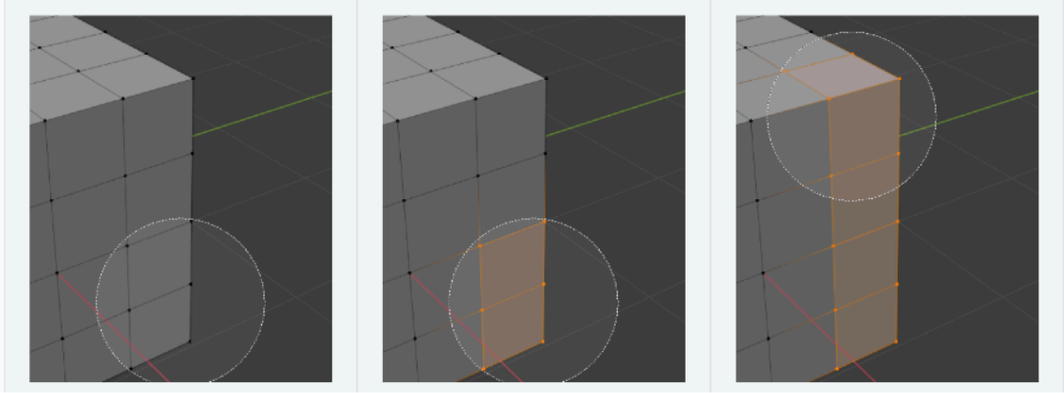
Meshes

Interactive geometry selection techniques

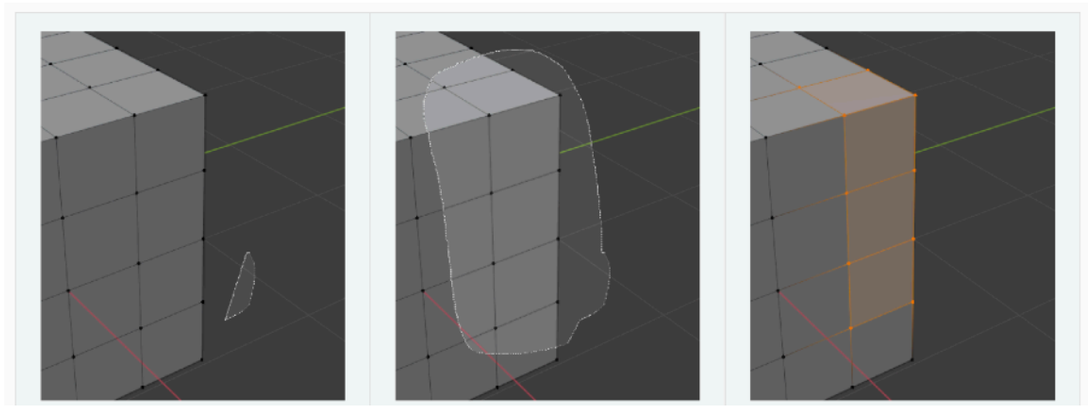
Box selection



Circle selection



Lasso selection

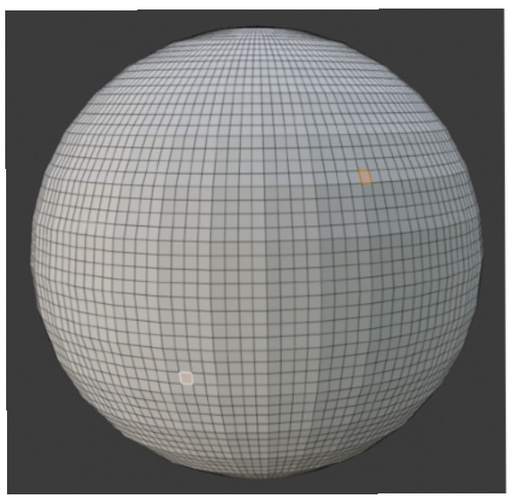


[https://docs.blender.org/manual/en/latest/interface/selecting.html#tool-select-box]

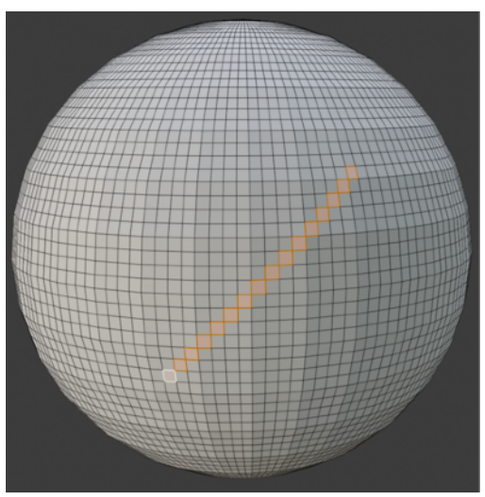
Meshes

Semi-automatic geometry selection techniques - I

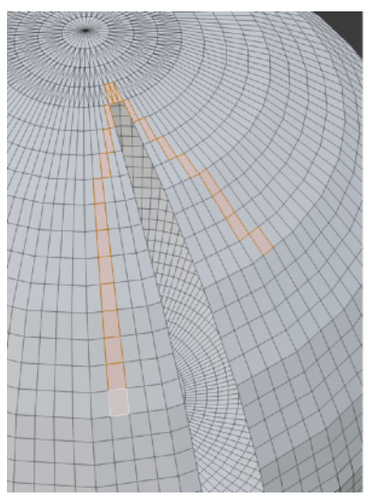
- Selection of the geometric elements that are along the **shortest path** (in terms of travelled length or number of traversed geometric elements) between two connected geometric elements
- It can be used sequentially to rapidly select a path of geometric elements **when edge/face loops are unable to express the animator's intents**, rather than selecting one by one
- In the **simplest case**, the path can be defined as the points of intersection between surface's edges and a **plane** that contains the two selected points and is parallel to their average normal
- For more **complex meshes** (with holes, obstacles) more complex **search algorithms** need to be applied (e.g., Dijkstra's algorithm), which are much more time consuming



Two user-selected faces in a mesh



Automatically selected faces across the shortest path between the two user-selected faces

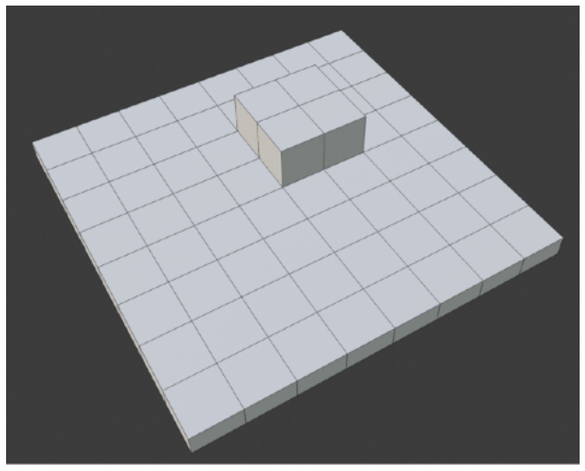


Result when there is no direct path between the user-selected faces

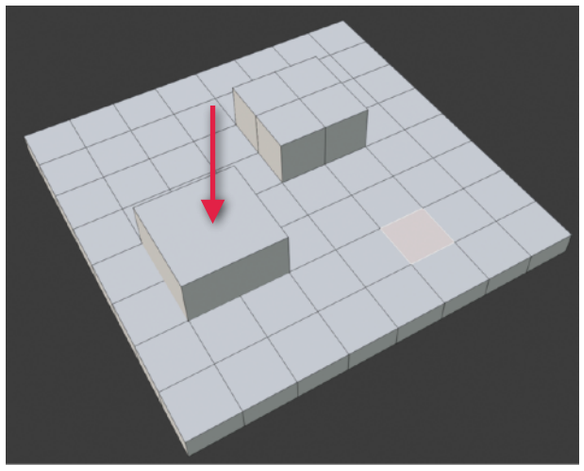
Meshes

Semi-automatic geometry selection techniques - II

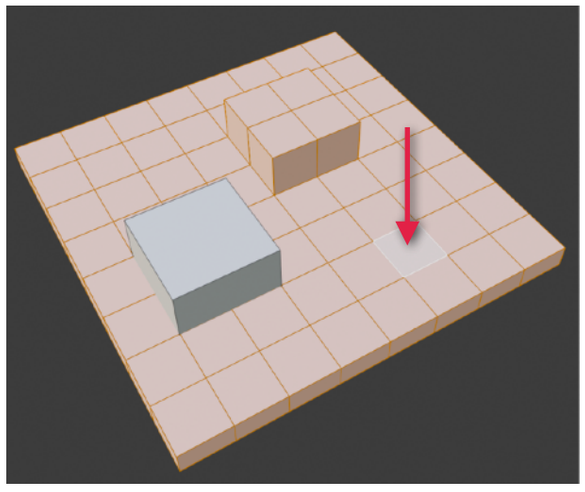
- Selection of all geometric elements that are **connected directly or indirectly** to a given geometric element, which is useful to **isolate elements that are disconnected** from all others;
- It is also possible to **label edges as *sharp*** to **constrain** the selection process among connected elements



A fully connected surface (water-tight)



A fully connected surface (water-tight) plus a disconnected cube overlaid (indicated with arrow)

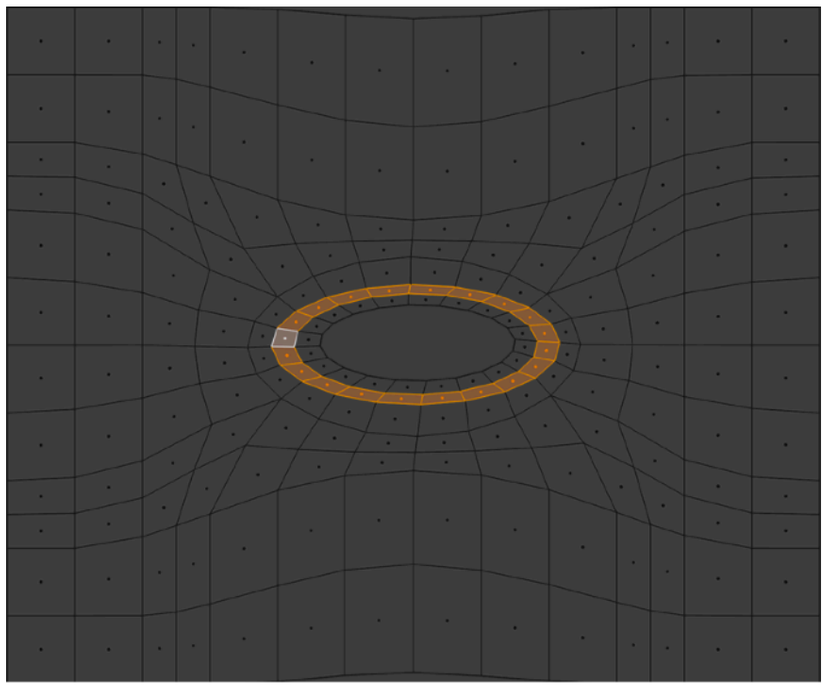


Automatically selected faces that are (in)directly connected to the user-provided face (indicated with arrow)

Meshes

Semi-automatic geometry selection techniques - III

- Selection of all elements present in **edge/face loops**, as we have seen in previous slides
- This selection method is useful as a preparatory step for **morphing organic meshes**, simulating the **effect of muscles**



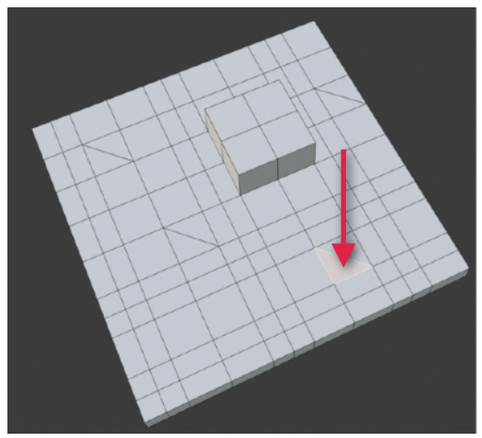
From a previous slide

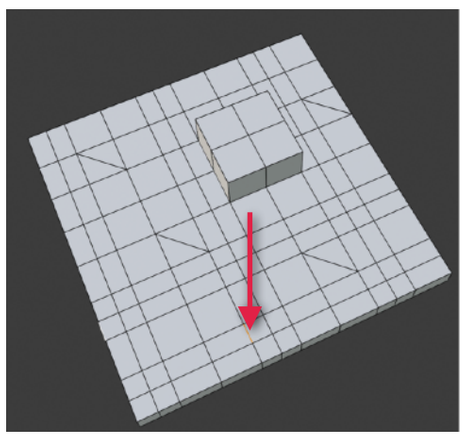
Meshes

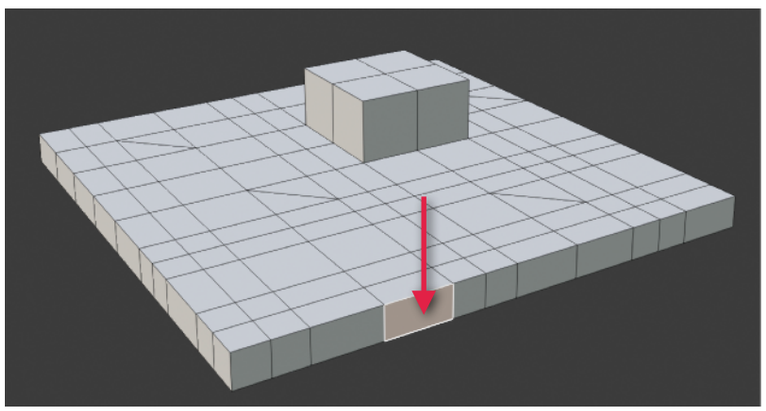
Semi-automatic geometry selection techniques - IV

- Selection of geometric elements that are **similar to a given geometric element**, not necessarily connected, in a given characteristic (acceptable similarity thresholds can be adjusted): **length**, **area**, **angles**, **local topology**, **material**, etc..

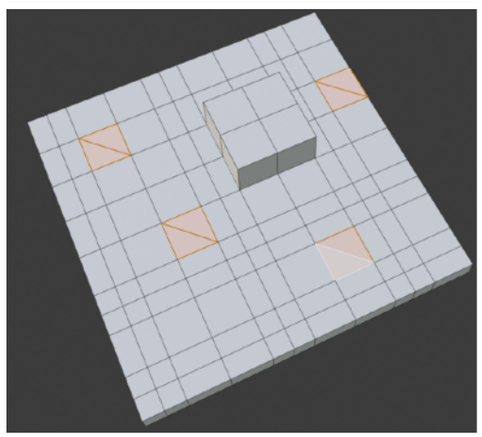
Mesh with user-selected geometric element (indicated with arrow)



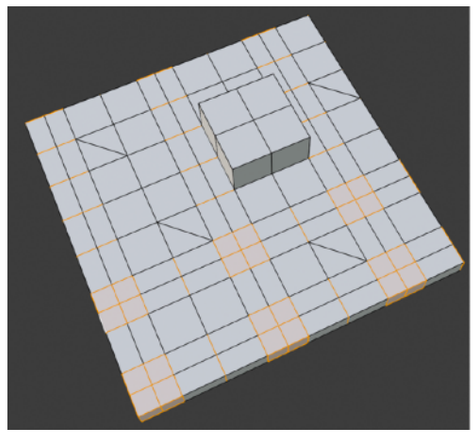




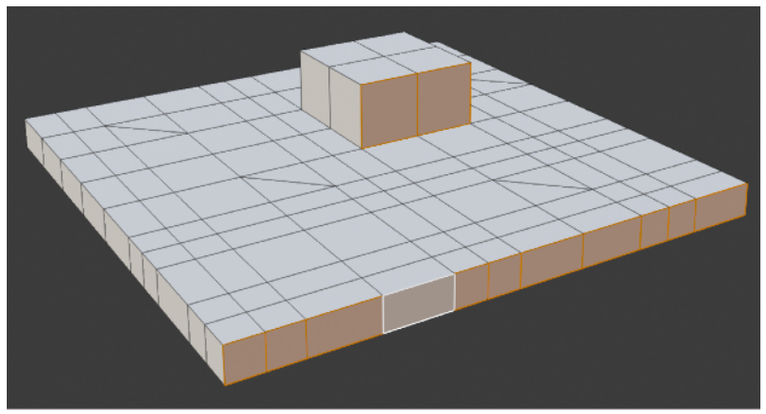
Automatically selected geometric elements given the user-selected one



Criterion: number of face's edges



Criterion: edge's length

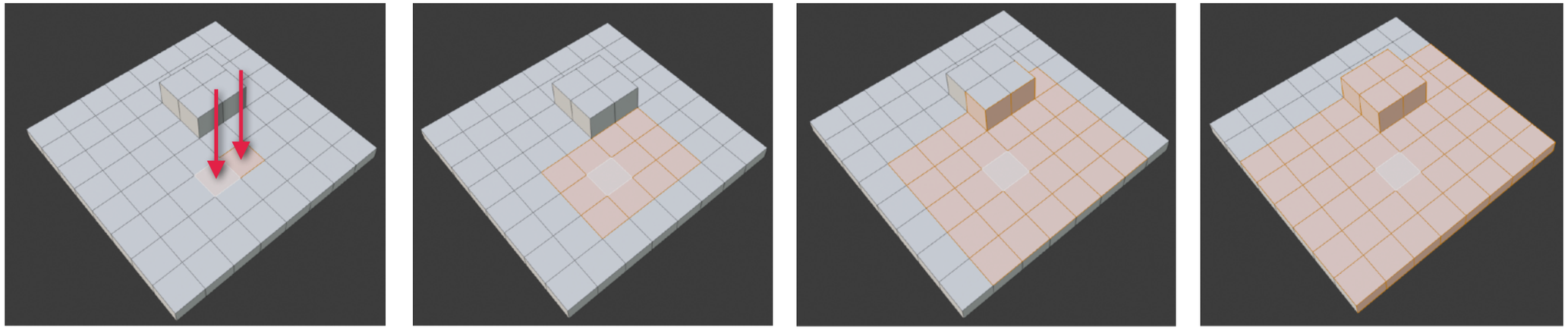


Criterion: normal's direction

Meshes

Semi-automatic geometry selection techniques - V

- **Incremental expansion/contraction** of an already existing selection to the nearest connected elements
- Selection method useful, e.g., to **dilate** the results obtained with **edge/face loops**.



Mesh with user-selected geometric elements (indicated with arrows)

Iteration 1

Iteration 2

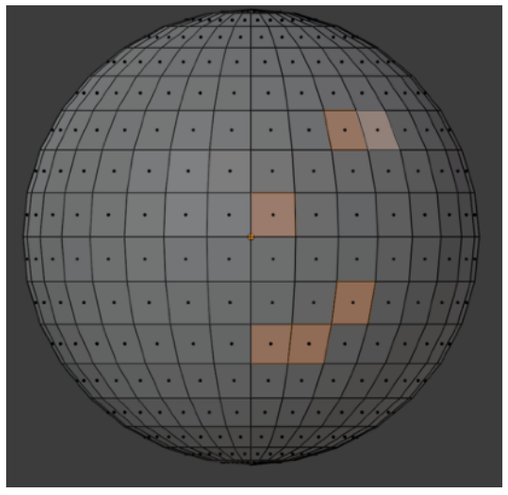
Iteration 2

Automatically selected geometric elements immediately connected to current selection

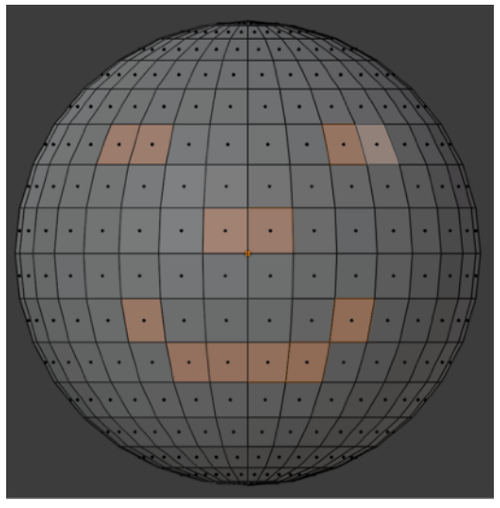
Meshes

Semi-automatic geometry selection techniques - VI

- Selecting geometric elements that are **exactly mirrored** with respect to a **given axis** of a given frame of reference (e.g., a frame aligned with the world and located at the object's centre)
- This functionality is particularly useful to model **symmetric objects**
- The dependency of the mirroring technique on the world frame requires the object to be centred and aligned with it for a proper operation



A few selected faces in a sphere, whose origin is located in its centre

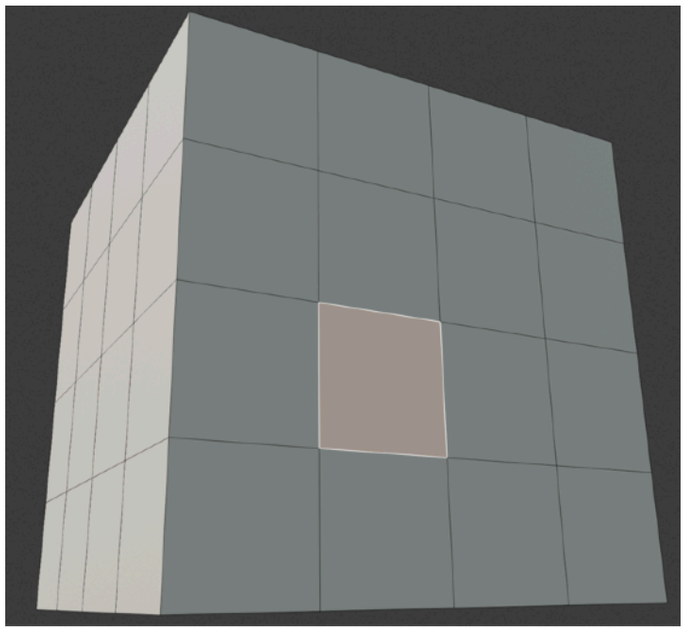


The result of selecting the faces mirrored w.r.t. to horizontal axis

The result of extruding and scaling the selected faces (w.r.t. to their origins)

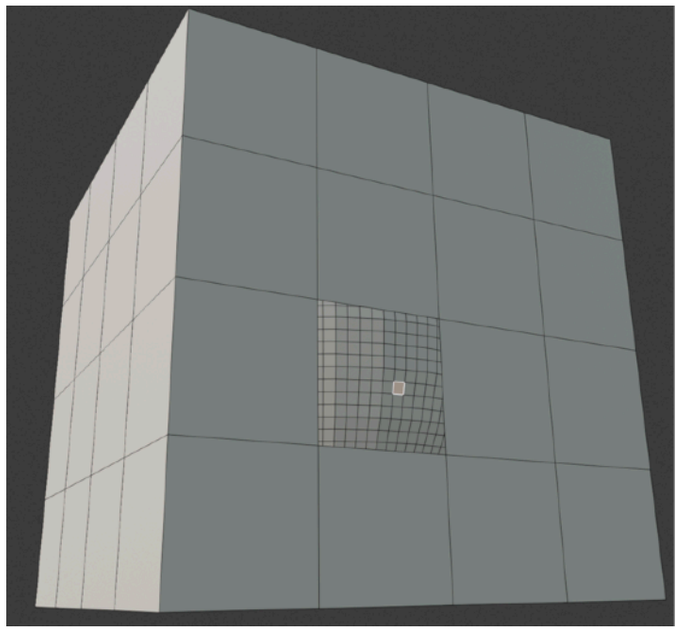
Meshes

Handling occluded geometry

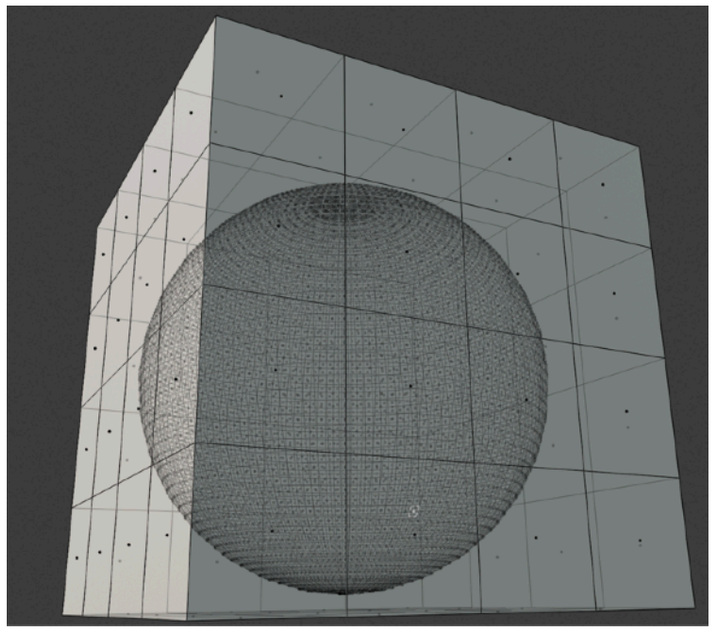


A cube mesh with a sphere mesh inside (**occluded**)

Geometry selection is limited to what is **visible** to the user



To gain access to the occluded geometry, we can **hide (not delete)** the occluding geometry

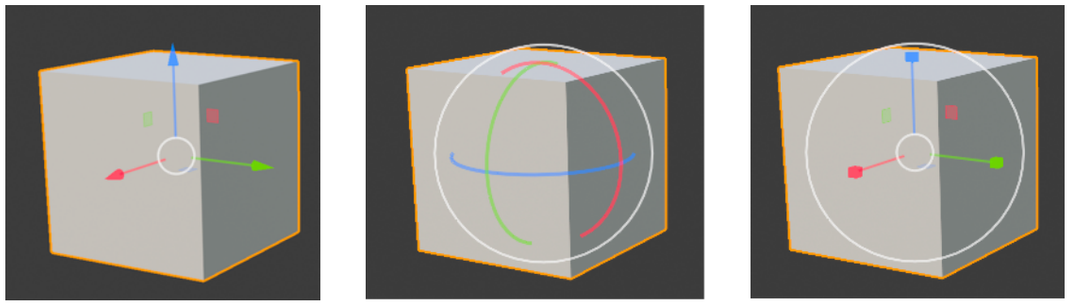


An alternative to hiding geometry is to activate what is known as ***x-ray*** vision, in which **opacity** is used to allow the user to select geometry that would be otherwise occluded

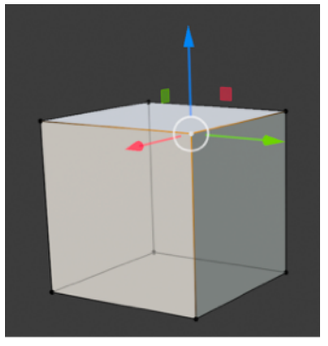
Meshes

Transformations - basics

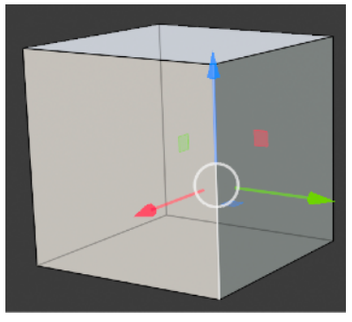
- **Rigidly transforming** (scale, rotation, and translation, by this order) meshes as a **whole** and its **constituents** (vertices, edges, faces) is a key operation
- 3D modelling tools offer mouse/keyboard controlled **visual tools** to perform these transformations, as well as **numerical sliders** for fine control
- These tools ensure that **transformations are applied consistently** (e.g., a user-applied translation to a face is internally processed as a translation applied to the face's vertices)
- ***Snapping*** (e.g., to grid, to other elements in the scene), ***axis locking***, and ***adaptive input*** *sensitivity (speed control)* are also typically available to overcome the difficulty of performing **accurate movements** with the input device (e.g., mouse)



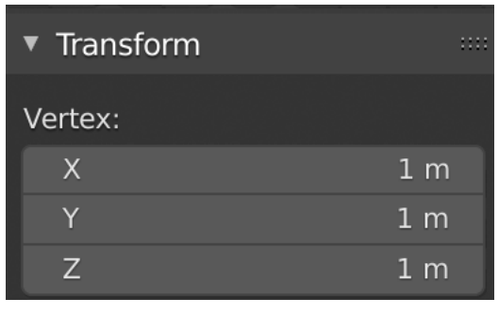
Translation, rotation, and scale visual tools for the whole mesh



Visual tool for vertex translation



Visual tool for Edge translation

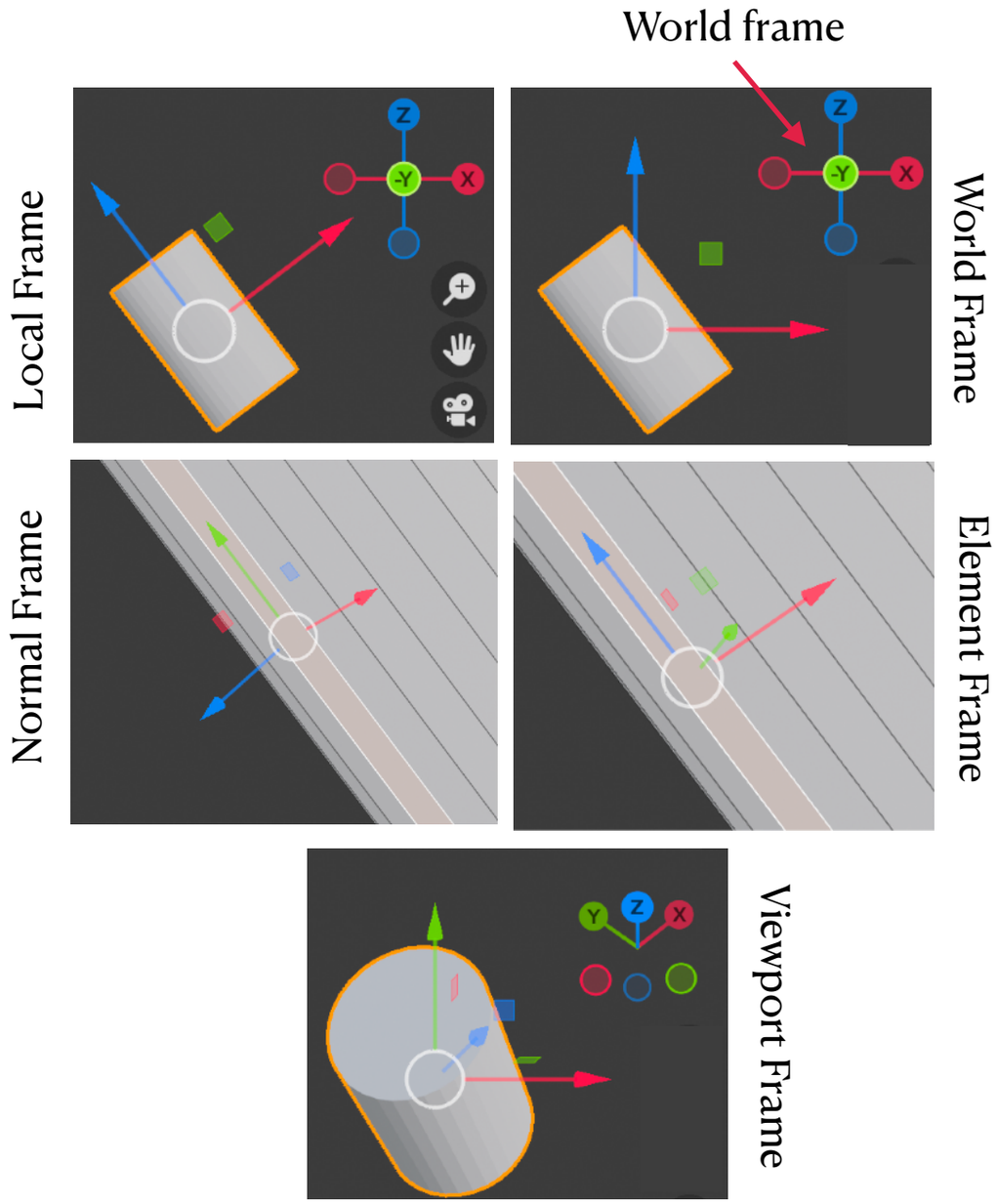


Numerical slider for translation control

Meshes

Transformations - frame of reference

- **Transformations are applied with respect to a given frame**, whose position and orientation can be defined by the user in many convenient ways (internally this involves calculation of **matrix products**)
- **Examples of convenient frames**:
  - **World frame**, which is fixed (useful for organising the objects to compose the scene)
  - **Object's local frame**, which rotates with the object (useful to transform and edit the object)
  - **Normal-aligned frame**, defined for each geometric element (useful for sculpting the object) or normal average if multiple elements were selected
  - **Viewport-aligned frame**, useful for transforming objects freely with the mouse
  - **User-defined frame** (aka ***3D cursor***), useful for all other cases the user finds convenient



Translation visual tool w.r.t. the user-selected frame (lateral labels)

Meshes

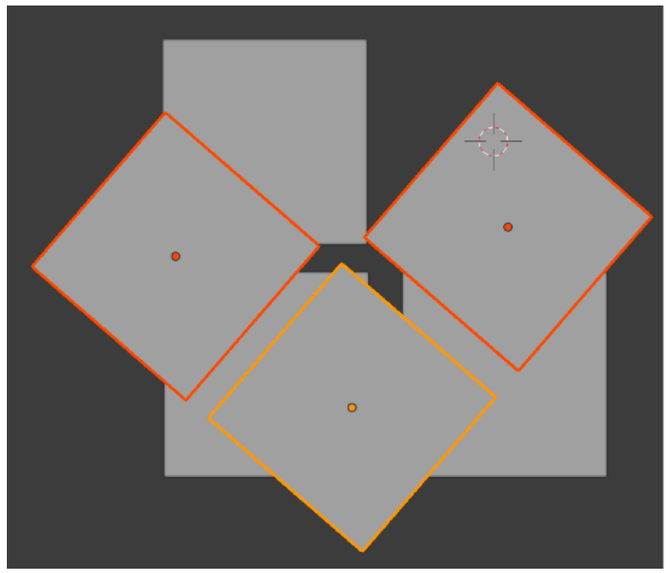
Transformations - pivot point

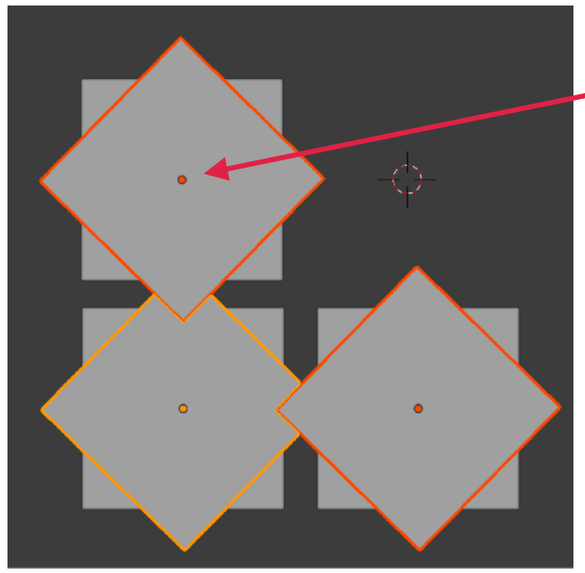
- Formally, **rotations** occur **around the origin** of the frame of reference in question
- Formally, **scaling** occur **towards/away the/from the origin** of the frame of reference in question
- When the frame is **on the object's centre**, the object rotates/scales **around itself** (as in previous slide)
- However, it may be convenient to rotate/scale **around any other user-defined point**, aka ***pivot point***
- To attain this, the **pivot point is used as frame of reference's origin**
- **Examples of useful *pivot points*** include:
  - The **object's origin**, which can itself be manipulated (e.g., snapping it to the 3D cursor)
  - The **centre of the bounding box** of the elements being manipulated
  - The **median point** of the the elements being manipulated
  - The **origin of any other pre-selected object**
  - The **origin of the 3D cursor**

Meshes

Transformations - pivot point

Pivot point placed at the median of the three cubes



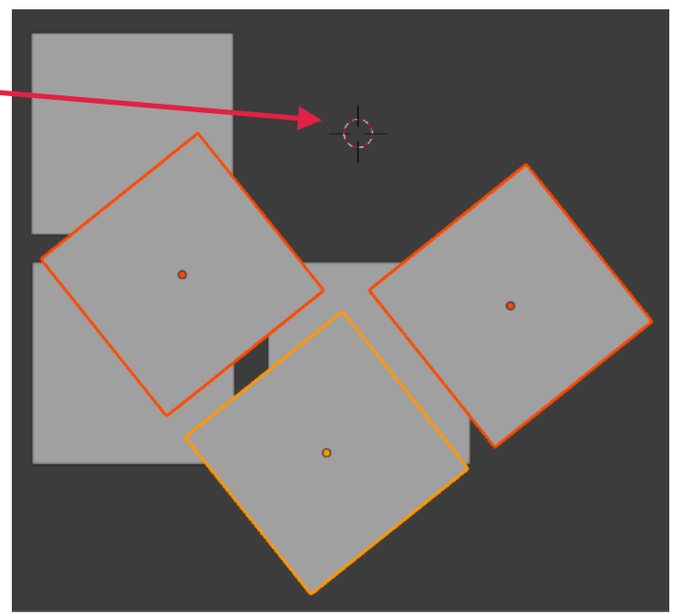


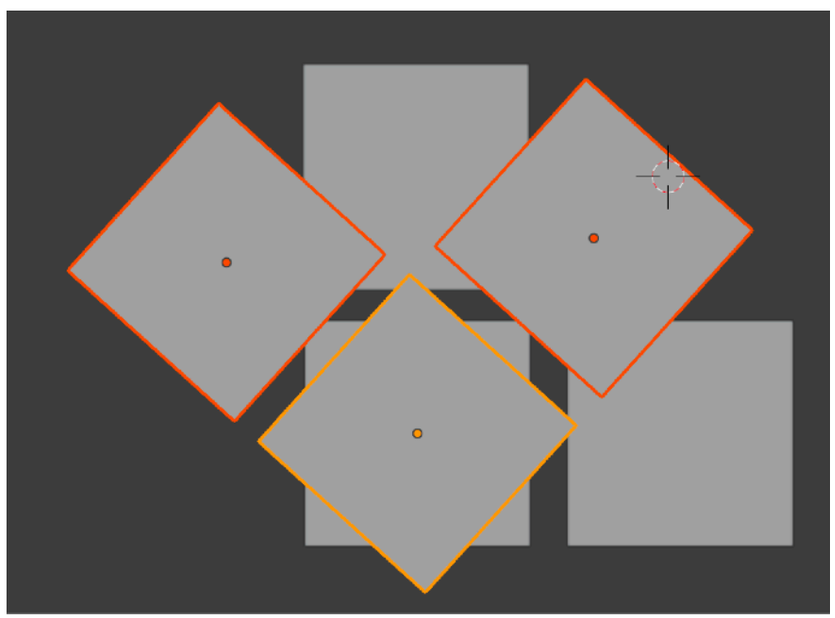
Origins represented as small dots

Pivot point placed at each cube's origin

3D cursor

Pivot point placed on the 3D cursor





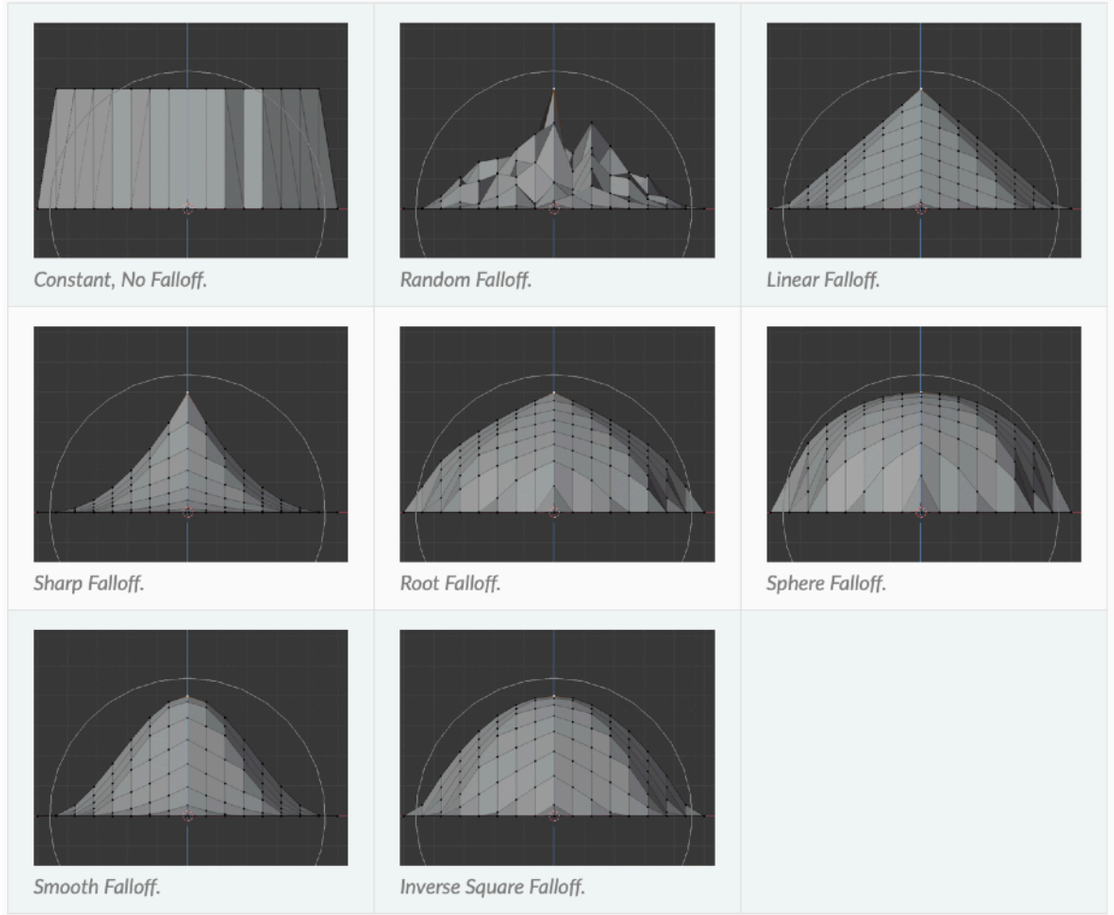
Pivot point placed on the origin of the selected cube (in yellow)

Meshes

Transformations - proportional editing

- **Transforming each element** (vertex, edge, face) individually is **time consuming** and hard to obtain a give **global effect**
- An alternative is to use ***sculpting tools***, whose purpose is to automatically influence the **neighbourhood** of the manipulated element
- Neighbours of the manipulated element are influenced if within a given ***radius of influence*** and according to a ***falloff function***
- *The **falloff*** function **returns a scalar** between 1 (at the manipulated point) and 0 (at the limit of the radius of influence) at a **rate** defined by a given mathematical function (e.g., constant, linear, inverse square, random)
- The transform applied to the manipulated element is then **scaled down** according to returned scalar before being applied to the **neighbour** in question

The effect of several falloff functions (the manipulated vertex is the central one)

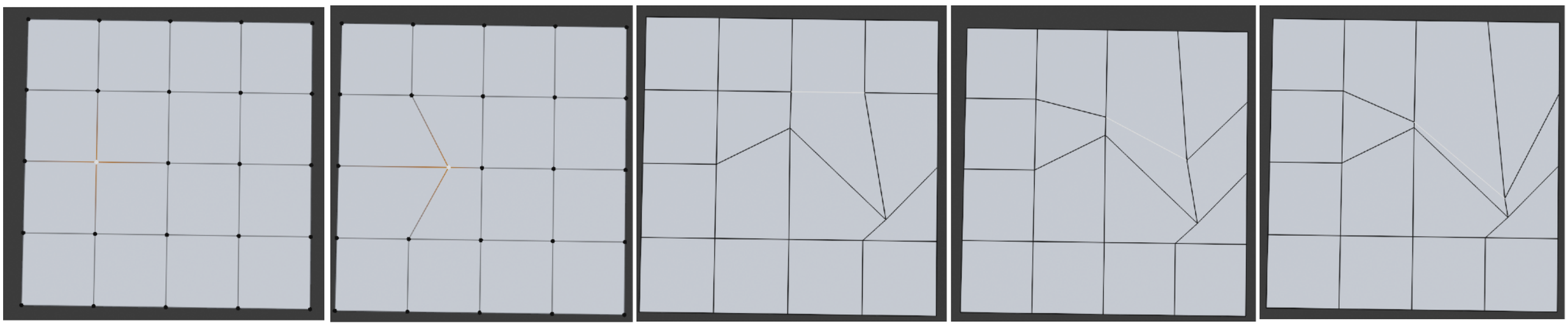


[https://docs.blender.org/manual/en/latest/editors/3dview/controls/proportional_editing.html?highlight=proportional%20editing]

Meshes

Transformations - constrained

- Vertices and edges can be **transformed constrained** to the geometry to which they belong, thus allowing the user to **refine the geometry** while maintaining its global properties
- The user can **slide vertices along the edges** to which they are connected, while maintaining the underlying topology
- The user can **slide edges (and edge loops) across adjacent edges**, in which the edges' vertices are displaced proportionally to the length of the adjacent edges (and edge loops)



A vertex selected

The vertex after sliding along a connected edge

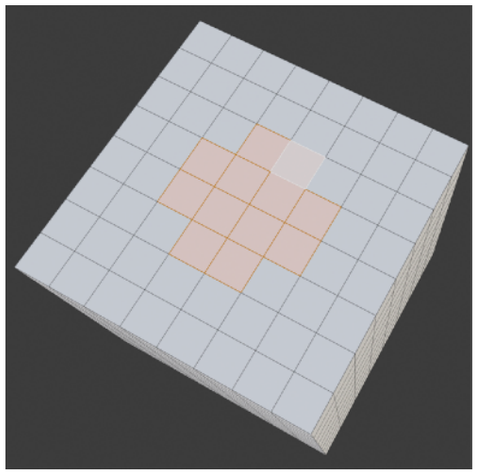
An edge selected

The edge after sliding across adjacent edges (40%)

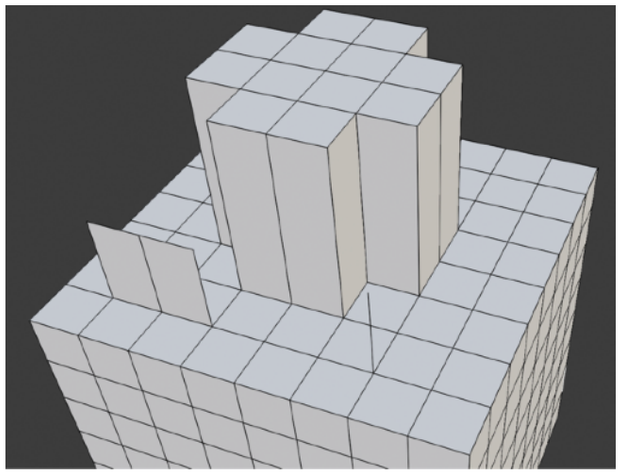
The edge after sliding across adjacent edges (90%)

Meshes

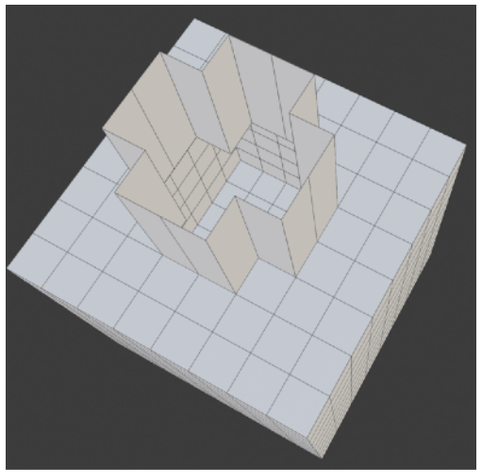
Adding geometry - Extrusion



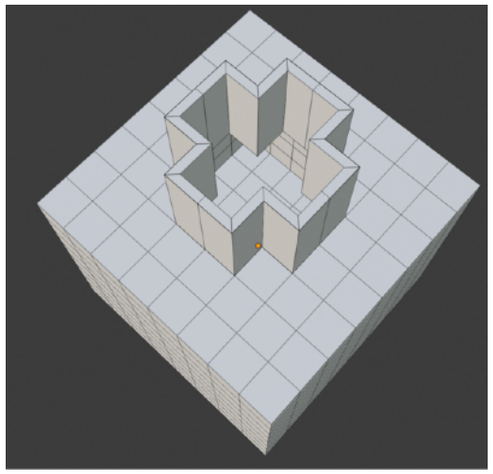
A set of faces selected for extrusion



The result of extruding the selected faces, plus two selected edges, and a single vertex



The extrusion result with the top faces hidden for inspection of the inner structure



The result of applying a *solidification* operation

- The operation of **extrusion** allows the animator to create new geometry from existing one; in a sense, it allows **adding "material" to the original shape**.
- The **vertexes and edges** at the boundary of the selection **are duplicated**, **moved**, and **connected** to the original ones to form new faces.
- Faces in selection's **inner region are simply moved** with the extruded elements (**not duplicated**)
- If faces are kept and then extruded **along their normals**, a ***solidification*** effect can be attained

Meshes

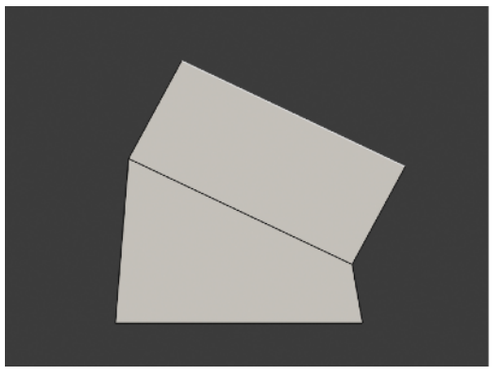
Adding geometry - Extrusion - Example

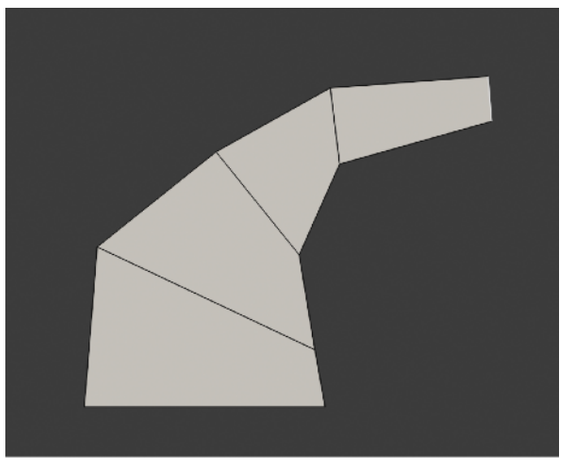
An edge

Extrusion

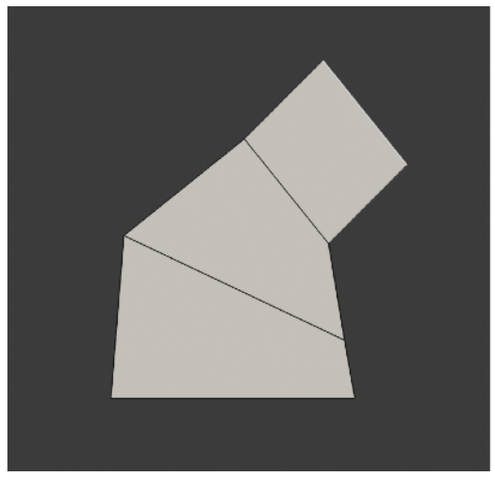
Rotation of extruded edge

New extrusion

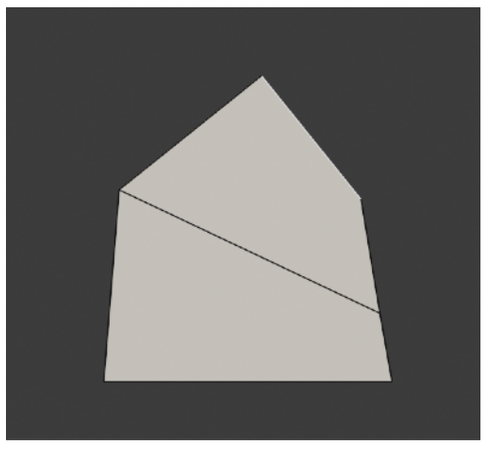




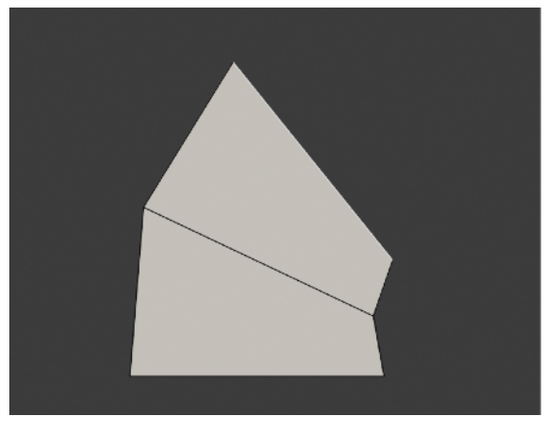
Scaling followed by rotation, extrusion, and scaling



Another extrusion



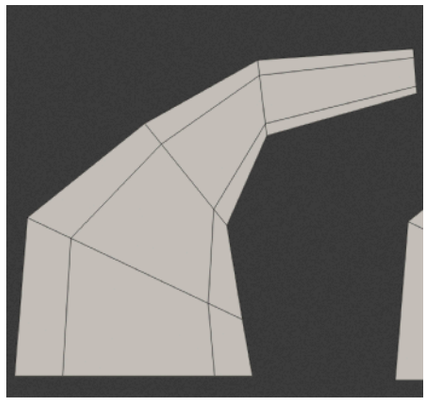
Scaling of newly extruded edge



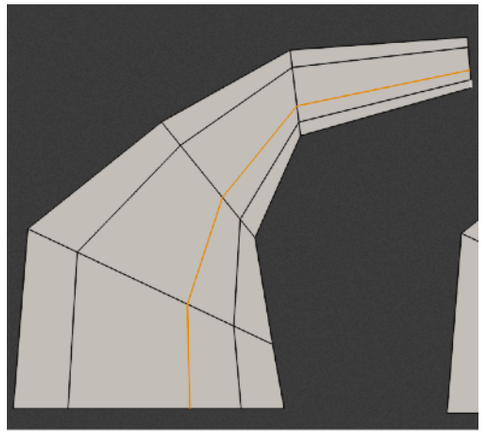
Rotation of newly extruded edge

Meshes

Adding geometry - Loop Cut

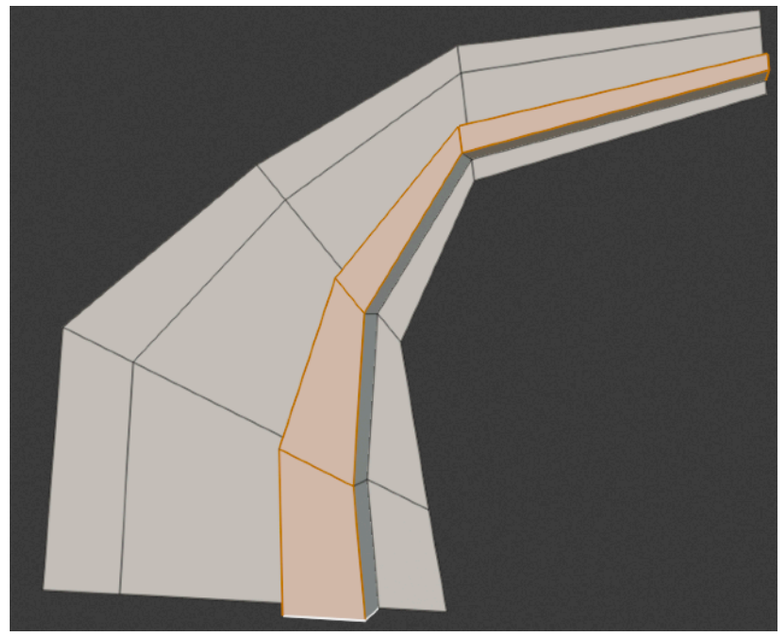


quad-mesh



A loop cut
(In orange)

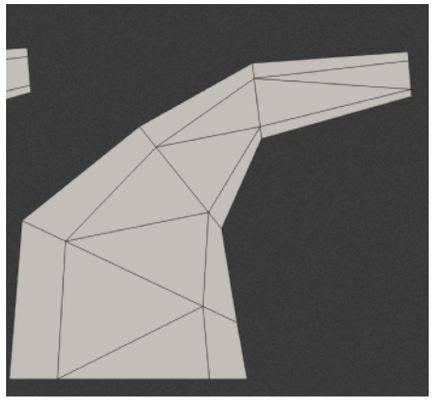
- Performing a **loop cut** means inserting an edge loop to **split a face loop**
- It can be used to **add detail** to the geometry, wherever it is needed (e.g., to help round it)
- Hence, it allows the animator to start **from a low-poly model and refine** it as required



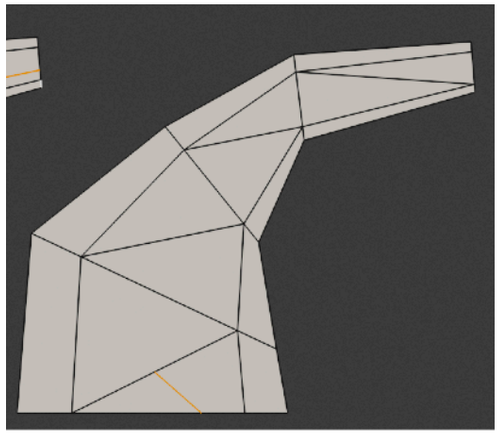
Face loop select followed by extrusion

Meshes

Adding geometry - Loop Cut



quad/tri-mesh



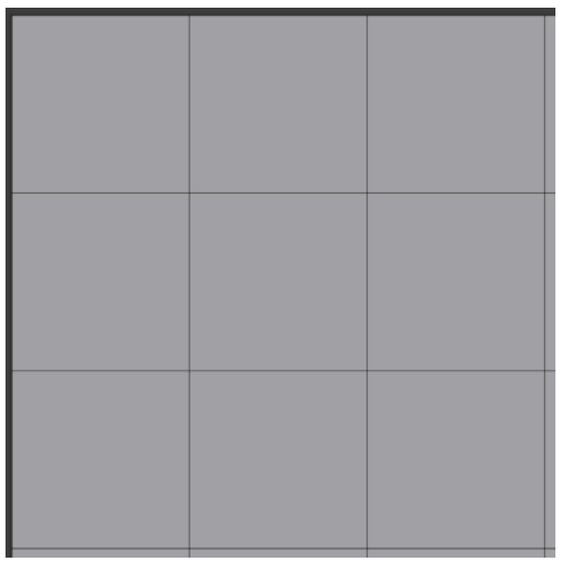
A loop cut
(In orange)

- Performing a **loop cut** means inserting an edge loop to **split a face loop**
- It can be used to **add detail** to the geometry, wherever it is needed (e.g., to help round it)
- Hence, it allows the animator to start **from a low-poly model and refine** it as required
- Again, it **depends** greatly on the **topological quality** of the existing mesh

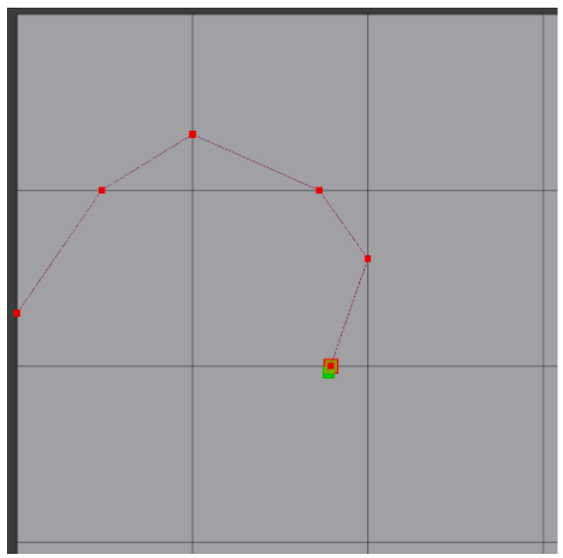
This loop cut does not meet the designer's intents due to poor topology (presence of triangular faces)

Meshes

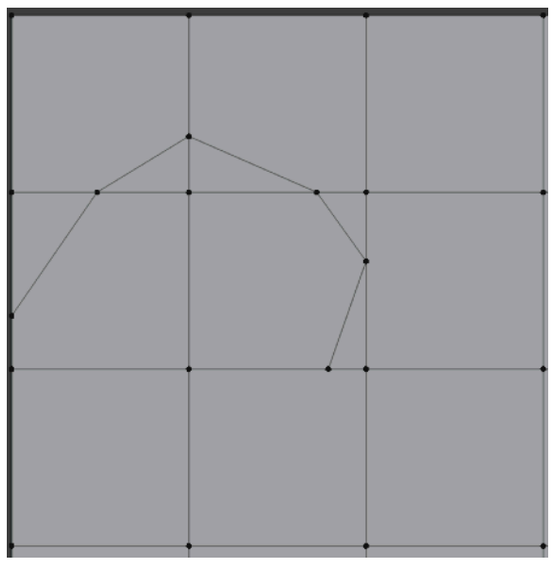
Adding geometry - Knife Cut



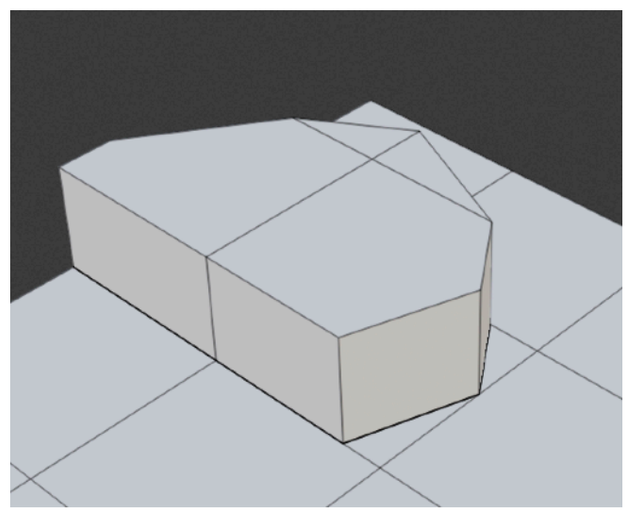
quad-mesh



cutting points in red



geometry after cut



Extrusion

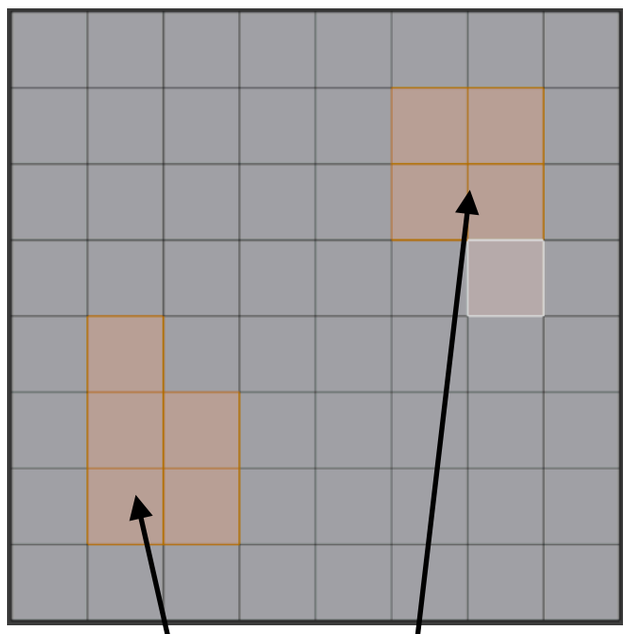
- Sometimes it is desirable to **cut** the existing geometry **without relying on face loops** or without doing it in a regular way (direct subdivision)

- The **knife operation** creates new user-defined vertices along existing edges and re-wires the mesh to include the new vertices ensuring the model remains “water tight”

Meshes

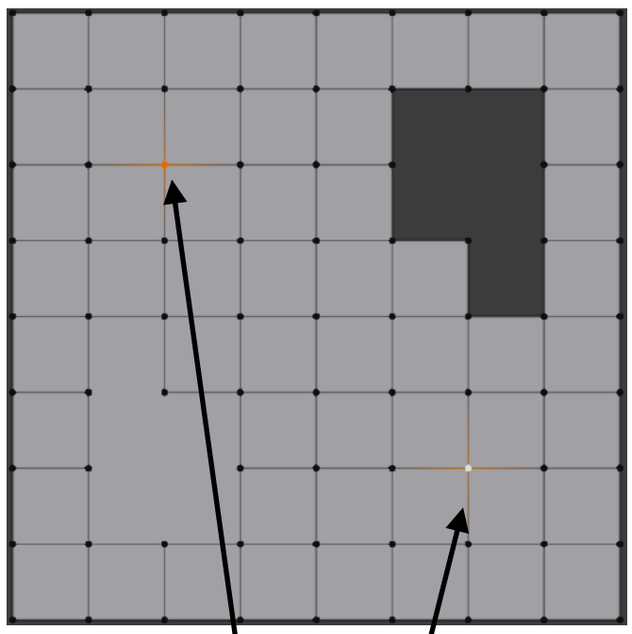
Removing geometry - delete vs dissolve

- **Eliminating** geometric elements in a mesh is often necessary, either when **(re)organising** the mesh or when **removing unnecessary detail** that what left from previous operations
- In some cases, instead of ***deleting*** geometry, the modeller opts for ***dissolving*** it, which means removing the geometry and then **fill in the hole**, usually in the simplest way possible
- ***Dissolving*** often generates polygons with more than 4 sides, **breaking edge/face loops**



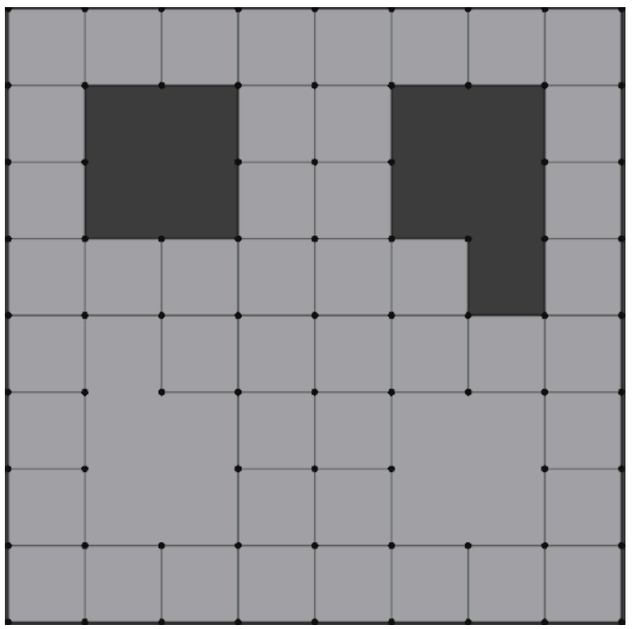
Area to be *dissolved*

Area to be *deleted*



Vertex to be deleted

Vertex to be dissolved



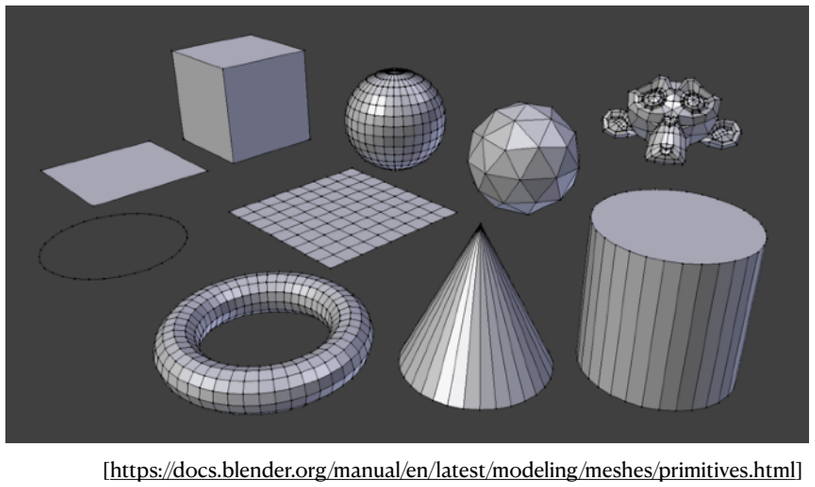
The final result

Meshes

Primitives

- Modelling often starts adding a pre-defined **parameterised** primitive generated **procedurally** on the current position of the 3D cursor
- Then, these primitives can be **edited/sculpted**, as if they were made of clay (just a metaphor, as many others)
- Primitives and their edited/sculpted versions can also be **combined** to create more complex figures
- Combining meshes via **parent-child relations** is **non-destructive** in the sense that the combination can be easily undone; however, their **geometric elements do not interact** directly (it may impact on morphing)
- The alternative is to ***concatenate/joining*** the meshes into a single one, which can be followed by some **bridging**/**merging**/**organising** operations to connect them; this can be a **destructive process** as undoing the joining process may generated unexpected results.

Examples of primitives



[https://docs.blender.org/manual/en/latest/modeling/meshes/primitives.html]

Bridging means connecting elements by adding faces to existing edges

Merging means collapsing a set of vertices into a single one, dissolving all the others

Organising means ensuring a proper topology to enable edge/face loops

Meshes

Parenting

- **Complex objects** are often more easily modelled by **individual parts** (objects themselves), which are then **hierarchically organised** in ***parent-child*** relationships to build the intended composition
- The *parent-child* relation ensures that **transformations** applied to a given part (scale, rotation, translation) **also affect the part's children**;
- Hence, **transforming the top-most** parent results is **transforming all parts** as a whole (e.g., moving the whole object in space), whereas translating a **leaf** in the hierarchy **does not affect any other part**
- **Example:** transforming the elbow changes the hand position, whereas rotating the hand does not affect the elbow's position

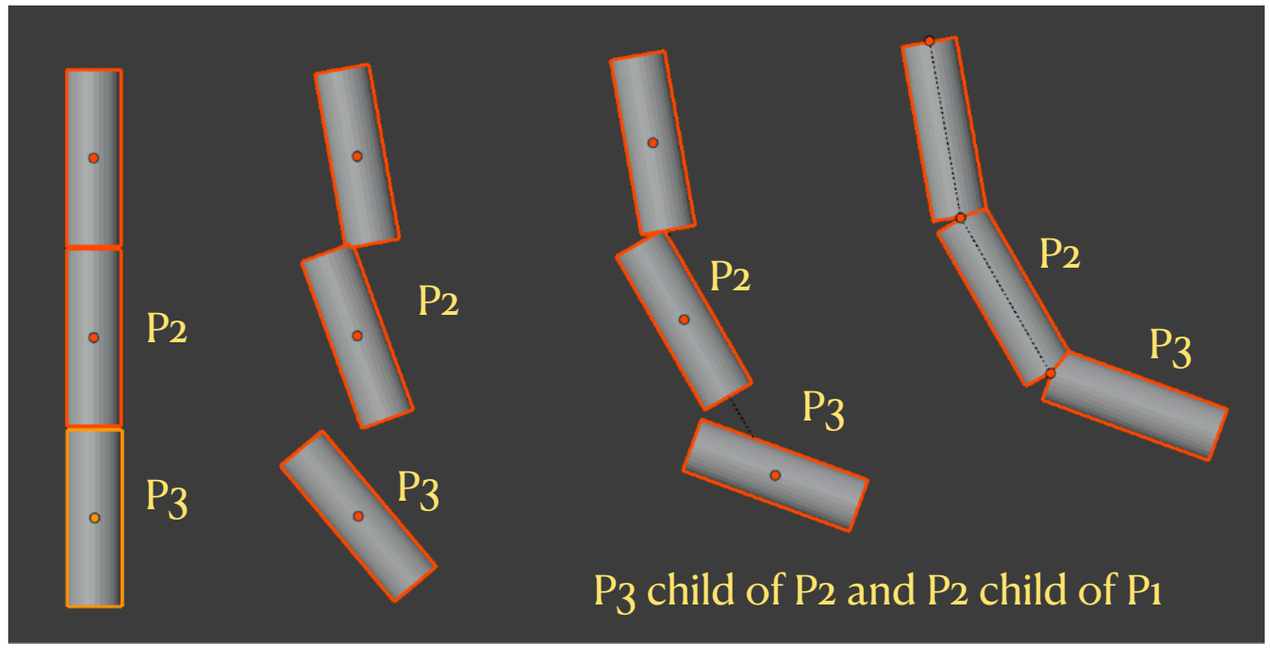
---

A: three parts in their original poses

B: without a hierarchy defined, transformations occur without cross-influence (in this case defined to happen around each part's origin - small circles)

C: with a hierarchy defined, transformations influence children

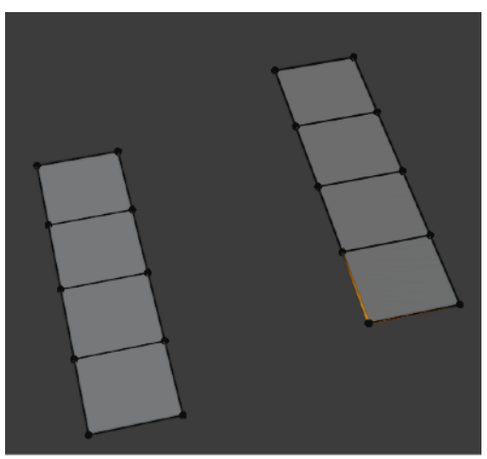
D: with a hierarchy defined and the parts' origins properly placed, transformations influence children in a more plausible way



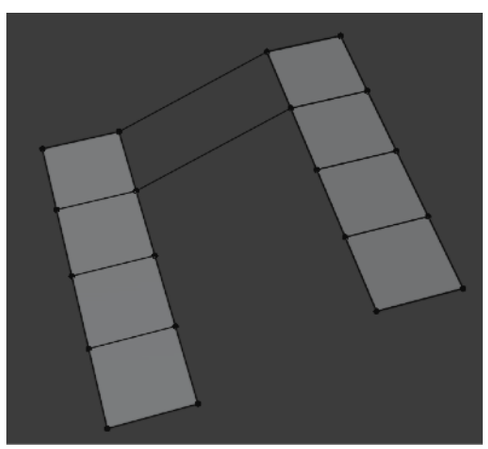
Meshes

Joining - Bridging - I

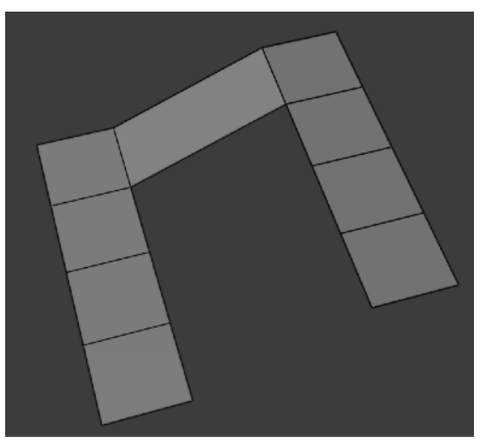
- When **merging meshes**, it is often necessary to ***bridge*** these by adding new geometry
- One approach is to: select two vertices in each mesh and **fill the space** between them **with an edge**; **repeat** this process for every pair of vertices required for the bridge; select pairs of newly created edges to **fill the space with faces**
- A faster approach is to **select two edge loops** in both meshes and ask for an **automatic process** that scans both loops while performing all required fill operations (edges and faces)
- The **faster approach requires both topologies to be organised** in a way that edge loops can be selected, scanned, and matched (the same number of edges)



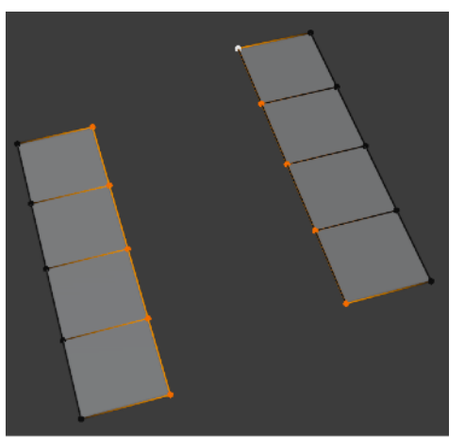
Two mesh segments needing to be *bridged*



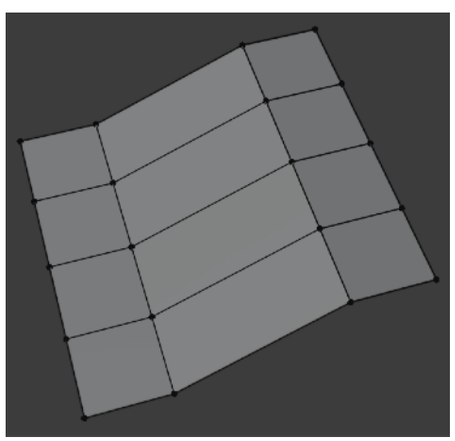
Two edges created from two pairs of vertices



A face created from four edges



Two edge loops selected

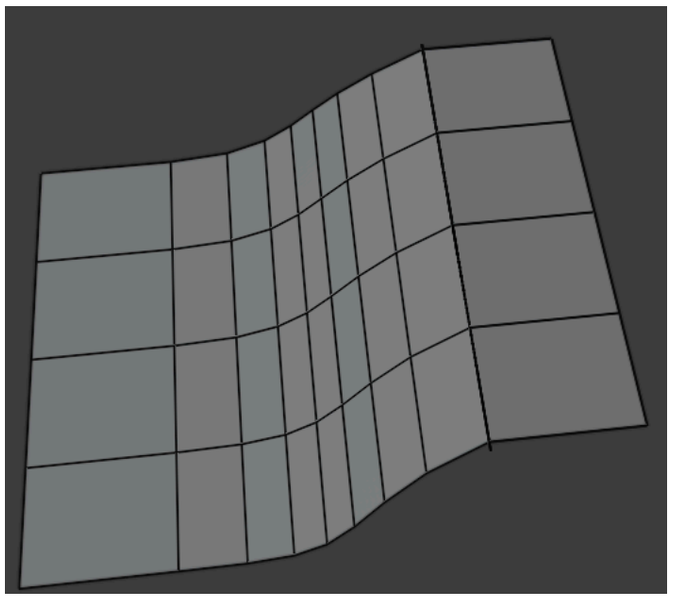


Automatic bridging from two edge loops

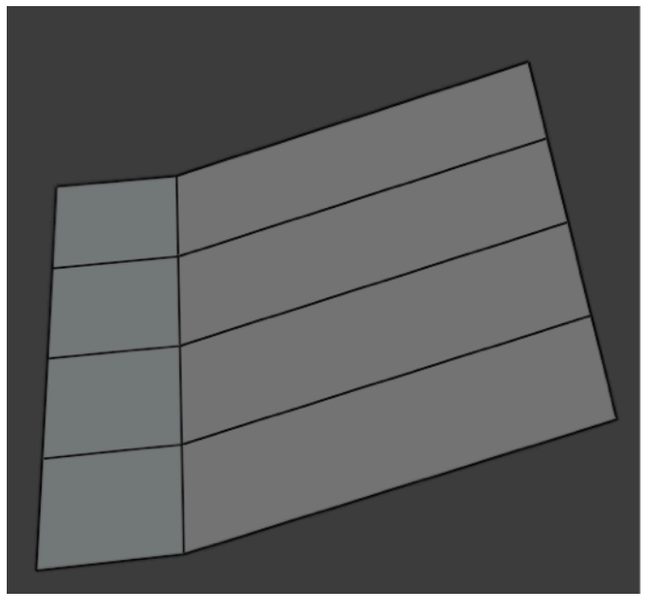
Meshes

Joining - Bridging - II

- Bridging can be **smoothed** by performing **intermediate loop cuts** and apply some **(non-)linear height equation** to all the intermediate loops
- **Alternatively**, edge loops can be **merged together**, as we have seen in the vertices case, which is only handy when both **meshes are really close to each other**



Bridging with intermediate loop cuts adjusted according to a non-linear height equation

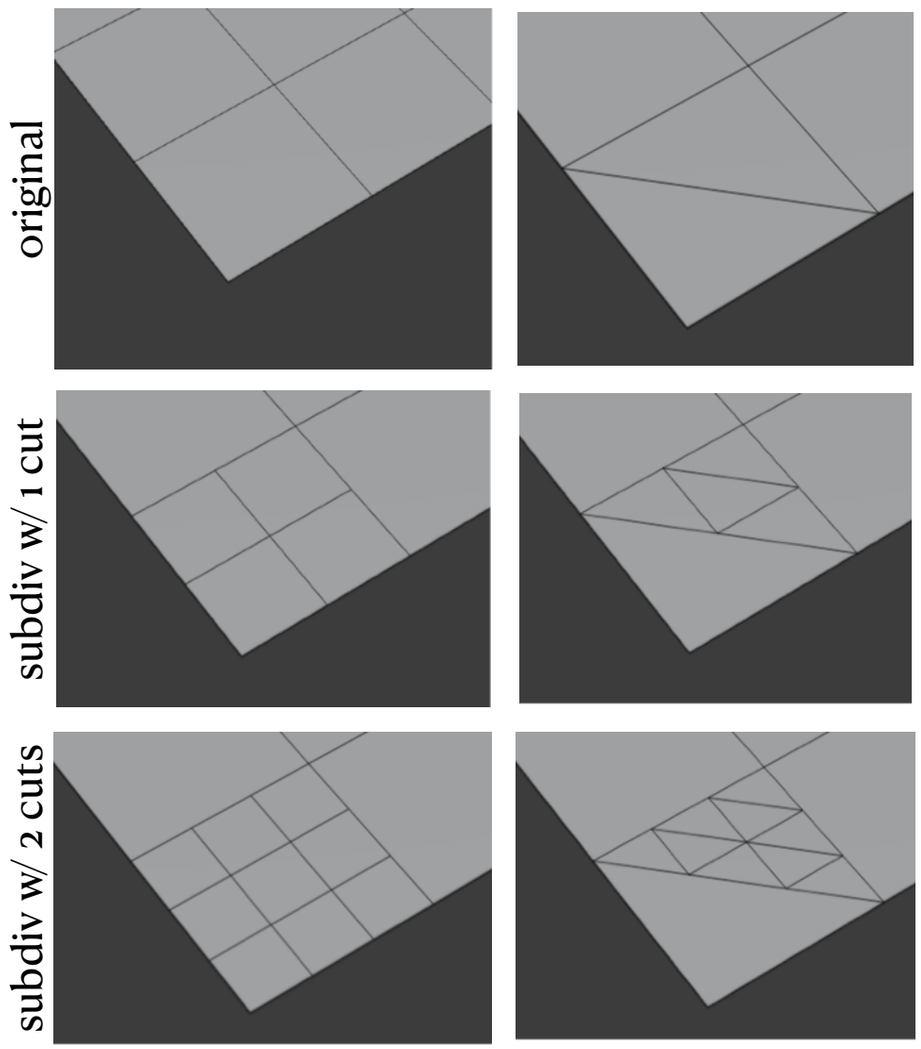


Merging edge loops (on the left one) instead of bridging them

Meshes

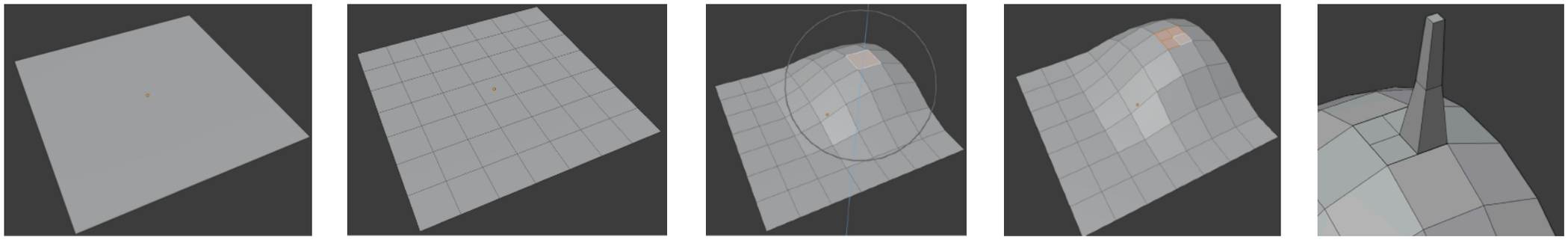
Adding detail - subdivision

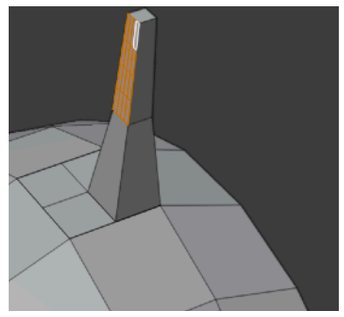
- When **editing** / **sculpting** meshes as if they were made of clay, it is a good practise to progressively increasing the mesh's **level of detail** on a **by-need basis**, instead of starting with a highly detailed mesh
- A ***coarse-to-fine*** or **top-down** approach helps de modeller progressively focusing from the overall shape to the fine details, while ensuring that the **polygon count** does not grow at large, which is key for **performance** issues
- Increasing mesh's localised level of detail can be attained with **surface sub-division** operations (we can subdivide faces and edges)
- In the case of tris/quads, a **simple sub-division approach** is to split the edges of the face in half (1-cut) or in more equal parts (n-cuts), which means placing new vertices in the cut points, and substitute the face by faces of equal area using the original, the new vertices, and additional ones added with simple geometric reasoning to hold the new surfaces (see figure).
- Many **variants** of this algorithm are able to produce different results, providing the modeller with **additional control**

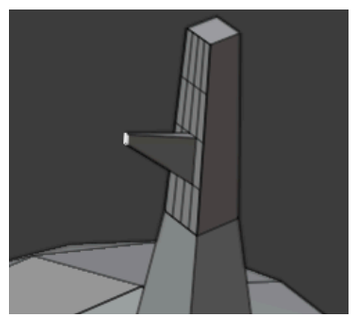


Meshes

Adding detail - subdivision - example





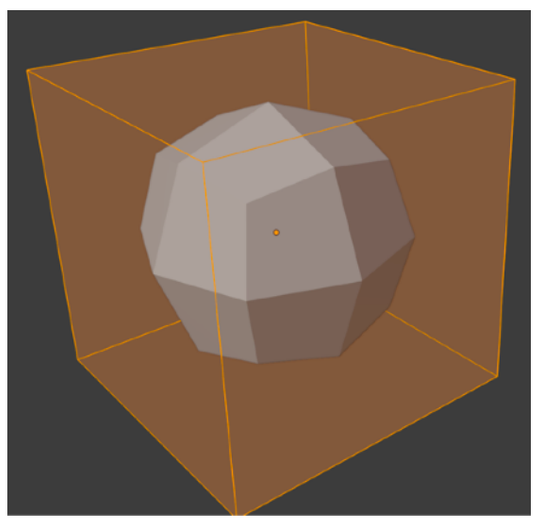


- This is an example of how to go from a single quad to a scene with an object using a **minimal number of geometry**
- **Subdivision** should be applied **only when needed**, and in this example is followed by extrusion (extending the quads), scaling (for thinning), and translation (to adjust position)
- In this example subdivision with **1 cut** is applied **recursively** until the desired **level of detail** is achieved.
- Note, however, that the result of these operations is a mesh with **several unnecessary polygons (useless detail)**, meaning that their **reorganisation** (e.g., dissolve) is required if a minimal quad-based mesh is to be maintained

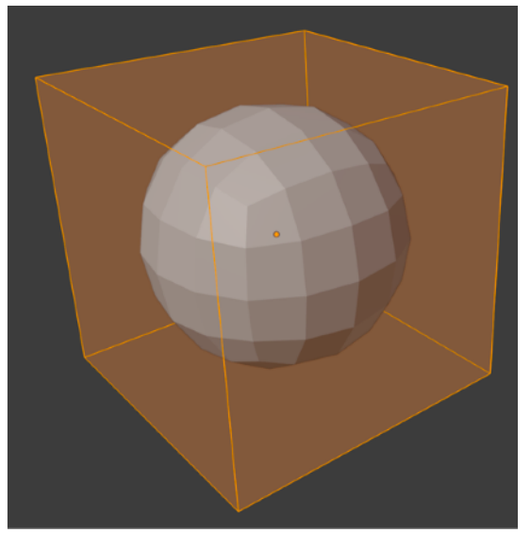
Meshes

Adding detail - subdivision with smoothing - example

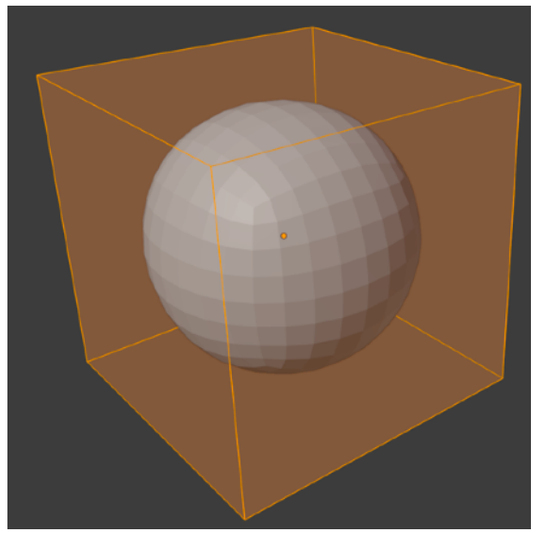
- **Catmull–Clark subdivision algorithm** (proposed in **1978**) is one of the most well known solutions for recursive smoothed subdivision of a mesh



1 iteration



2 iterations



3 iterations

In these figures, the grey geometry is the result of applying the Catmull–Clark subdivision algorithm to the orange cube

Catmull, E.; Clark, J. (1978). "Recursively generated B-spline surfaces on arbitrary topological meshes". Computer-Aided Design. 10 (6): 350. doi:10.1016/0010-4485(78)90110-0.

Meshes

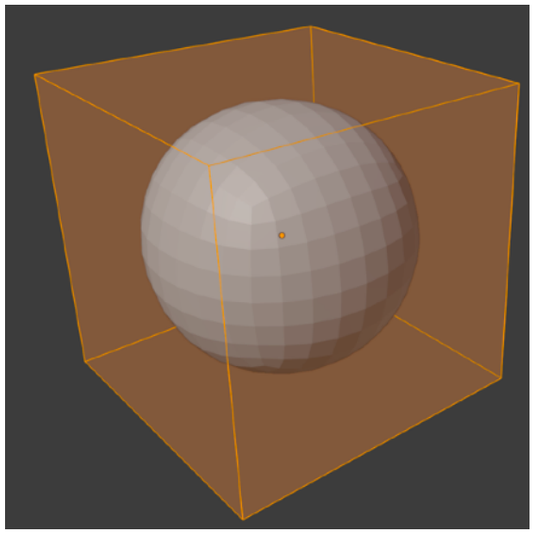
Adding detail - subdivision with smoothing - control

- **Catmull–Clark subdivision algorithm** (proposed in **1978**) is one of the most well known solutions for recursive smoothed subdivision of a mesh

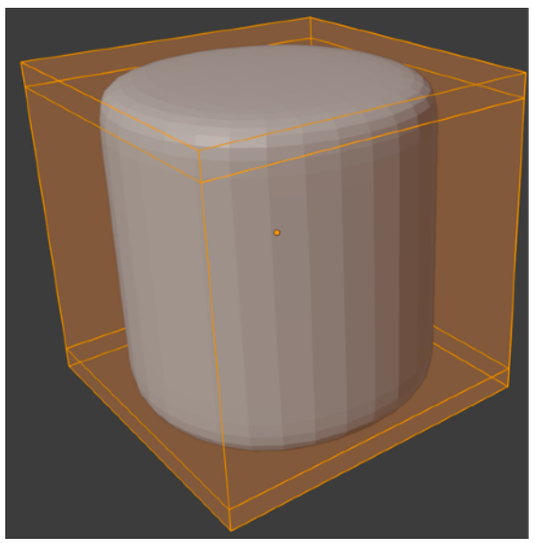
  - The **level of smoothness** can be **locally controlled** by altering the original geometry before applying the subdivision algorithm

  - Often **loop cuts** are employed for this purpose (see in the Fig. the two *loop cuts* that were added to the original geometry before the subdivision)

  - This reinforces the need for ensuring a **proper topology** of the models being generated; a bad early design will have significant repercussions



Result obtained with Catmull–Clark without previously applying loop cuts



Result obtained with Catmull–Clark after applying two loop cuts

Catmull, E.; Clark, J. (1978). "Recursively generated B-spline surfaces on arbitrary topological meshes". Computer-Aided Design. 10 (6): 350. doi:10.1016/0010-4485(78)90110-0.

Meshes

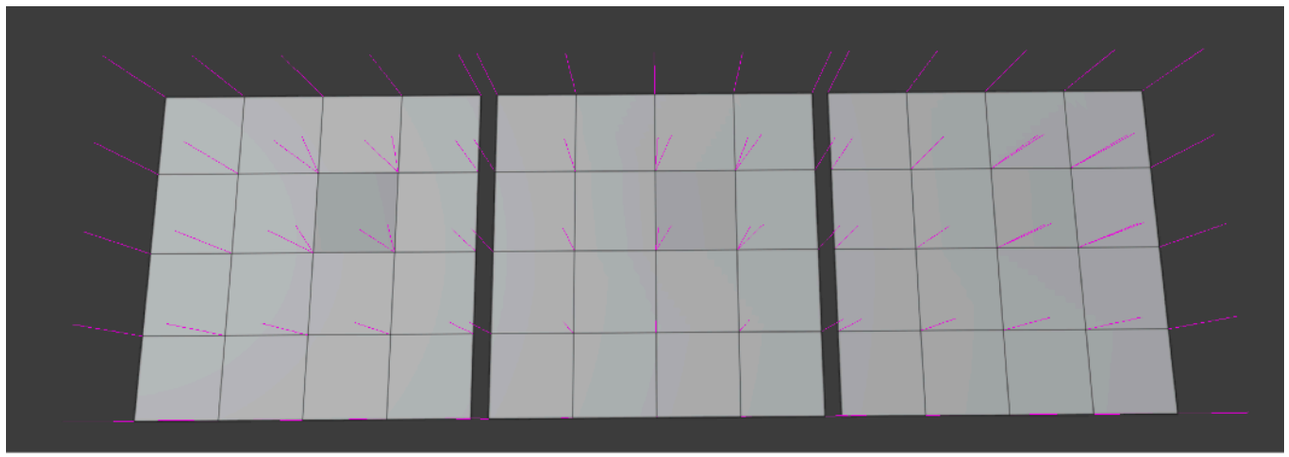
Normals - cleaning

- **Normal interpolation** (not transformation) is a temporary process executed at rendering; however, some **other processes** transform the **normal vectors**, hence, with non-temporary impact
- For example, **editing** and **importing normals** may result in **flipped and noisy normals**, which often result in substantial shading artefacts, thus requiring some cleanup
- Correcting ***flipped*** **normals** can be done **manually** (just reverse the normals directions) or it can be done automatically by determining which direction is the **most plausible/consistent**, given the normal directions of the connected geometry (e.g., choose the normal direction that minimises the average angle with the normals of the neighbour vertices)
- **Noise** in a given normal can be **filtered** out by **averaging** the normal vectors within a given **radius of influence** (connected geometry), leaving out geometry connected via ***sharp*** **edges** (manually or automatically labelled as such)

Left: a quad with manually rotated normals
(see the shading artefact)

Middle: the mesh after one smoothing iteration
(see the shading artefact disappearing)

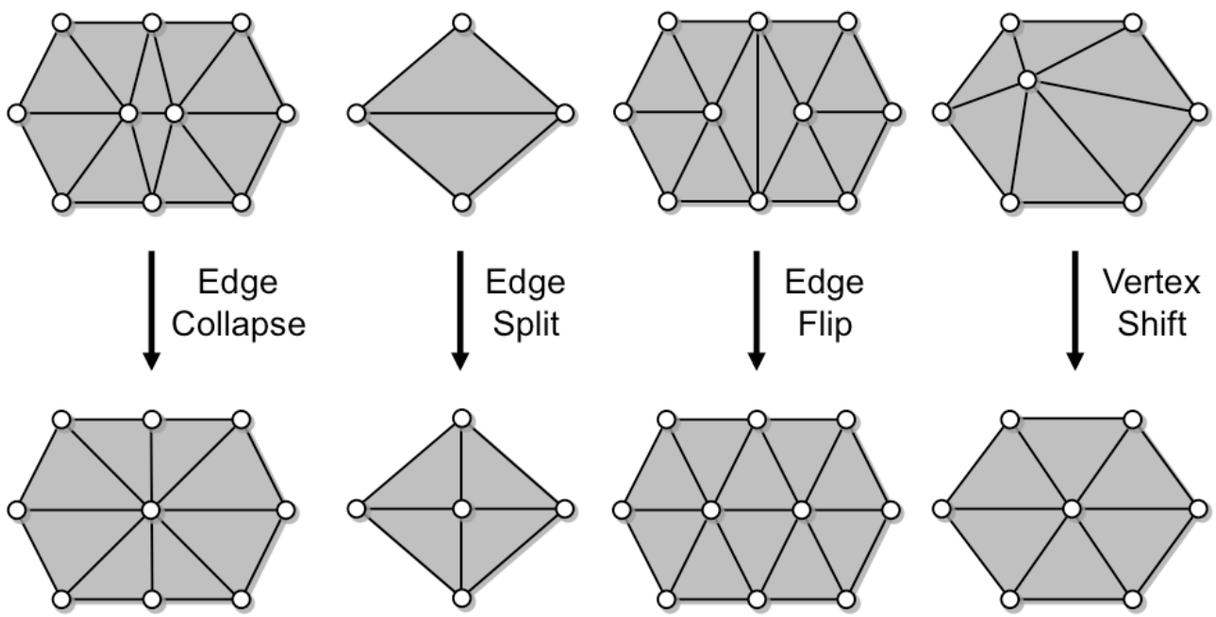
Right: the mesh after two smoothing iterations
(see the shading artefact barely present)



Meshes

Cleaning - re-meshing

- As seen in previous slides, a proper **mesh organisation** is key to **facilitate editing** (using loops) and **shading** (ensuring homogeneous texture sampling and interpolation)
- **Re-meshing** operations can be performed to **improve homogeneity** of the mesh (see figure as an example for the case of a triangular mesh)
- Re-meshing operations include: **edge collapsing** (vertex merging at centre or old vertex location), **edge splitting** (adding a vertex at midpoint of edge), **edge flipping** and **vertex shifting** (translation towards the average position of connected vertices)

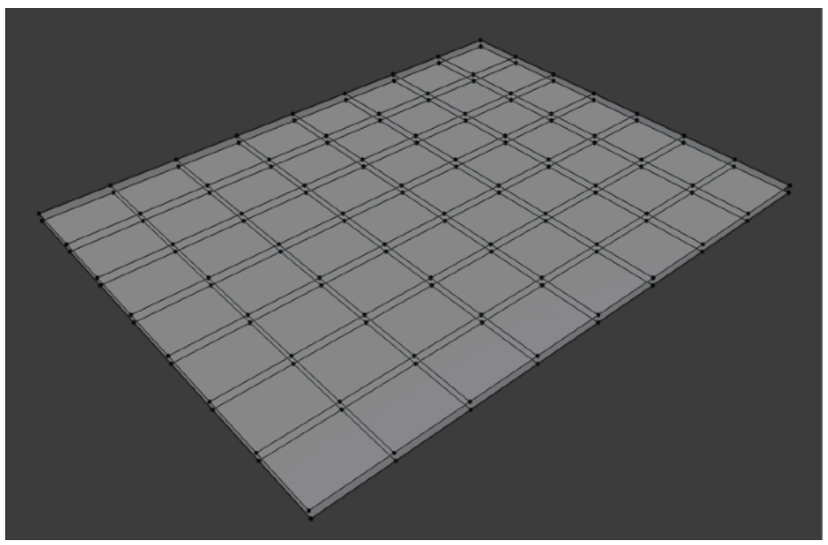


[Bickel, 2011]

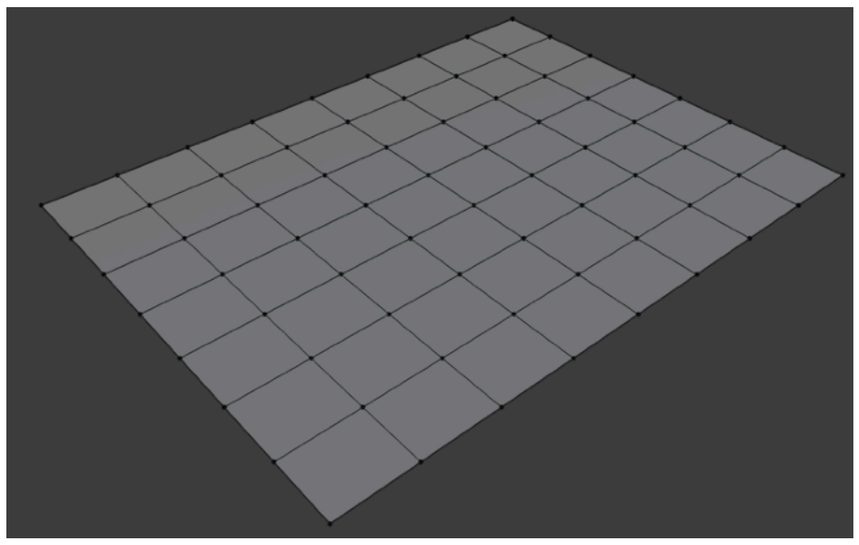
Meshes

Cleaning - redundancy, degenerate, and loose

- **After editing** a mesh, it is often the case that geometry needs to be **merged by distance**, otherwise problems will arise when further editing, animating, and shading
- **Merging by distance** operates at the vertex level, in which **vertices** that are **closer to each other** than a given user-defined threshold are **merged** (one is deleted) and edges/faces are **rewired** accordingly
- **Loose geometry** (unconnected vertices, edges, and faces) can be **deleted**
- **Degenerate geometry** (e.g., no-length edges, no-area faces) can be **dissolved**
- **Normals** can be **recalculated** to fix any problems introduced by geometry manipulation operations



Before merging: 144 vertices / 112 faces



After merging: 72 vertices / 56 faces

Meshes

Procedural modelling tools

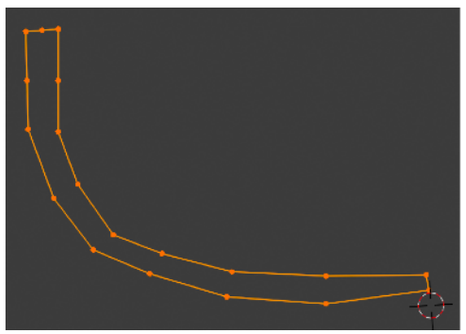
- To **speed up** their work, modellers recur to **procedural (algorithmic) tools** to rapidly build structures, which can then be **tuned** according to style/aesthetics intents

Meshes

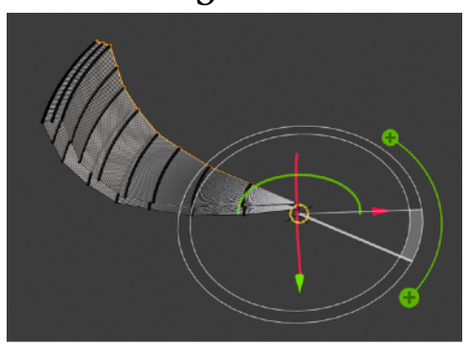
Procedural modelling tools - Spin

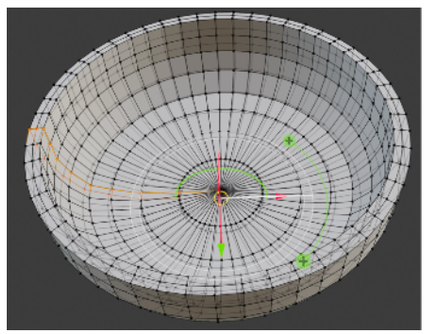
- An example of a **procedural tool** is the one that **extrudes repeatedly** (number of repetitions defined by the user) a given **geometry** around a **central point** (3D cursor), which is useful to build detailed **isotropic** objects

Geometry to be replicated (Only vertices and edges)



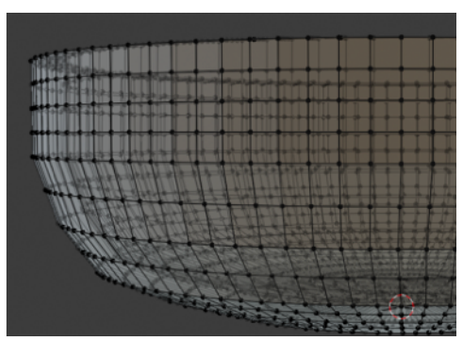
First replications around 3D cursor

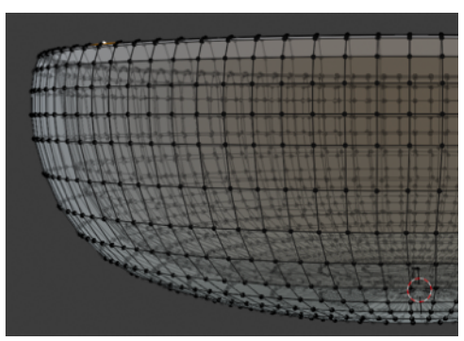




Full replications with vertex merged by distance in the last one to ensure a *water tight* model

A side view of the generated geometry shows that it isn't as smooth as desired





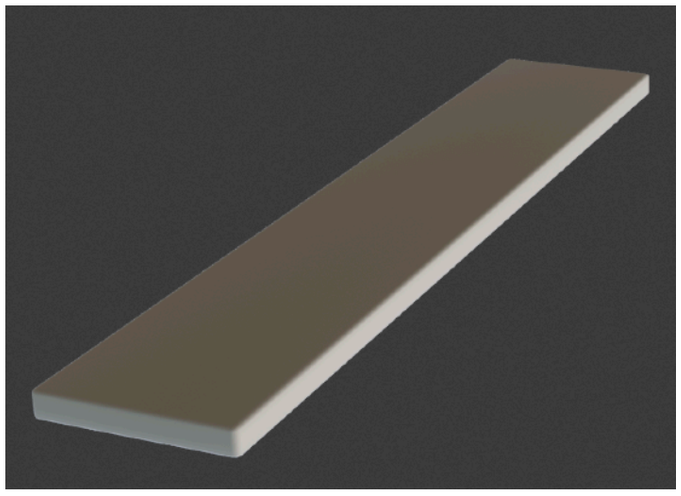
After a few Laplacian smoothing iterations

The final rendering with smooth shading activated

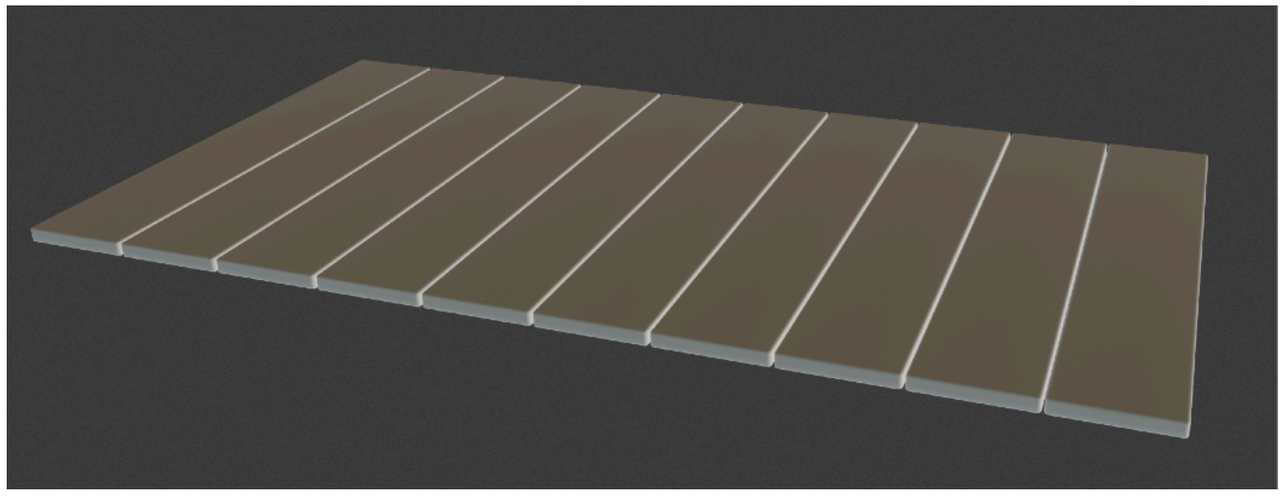
Meshes

Procedural modelling tools - Array

- **Repetition** can also occur across a given **path**, in which case the user needs to set the path, the number of replications
- **Repetitions** can be **merged** together or they can have some **offset** between them, defined by the user



Geometry to be replicated: a scaled cube with bevelled edges

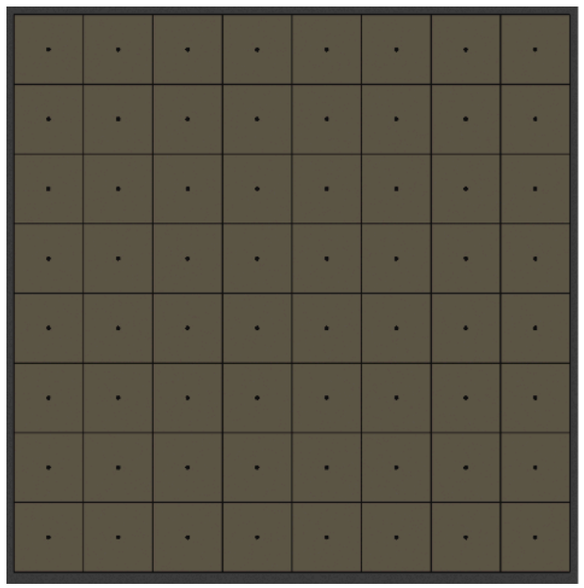


Replication parameterised for 10 items with some offset between them

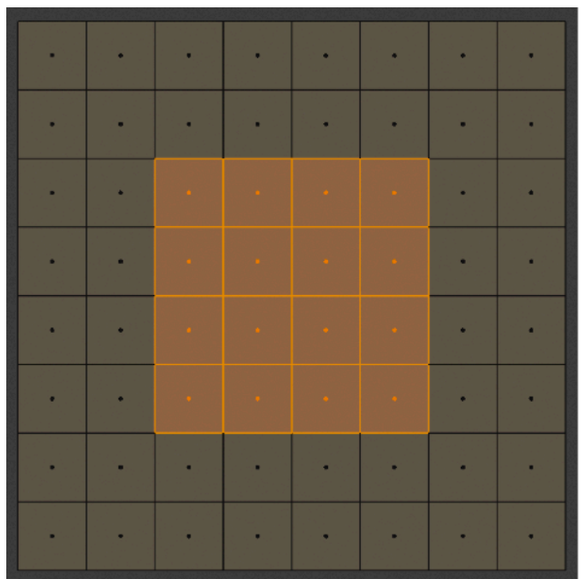
Meshes

Procedural modelling tools - model fitting

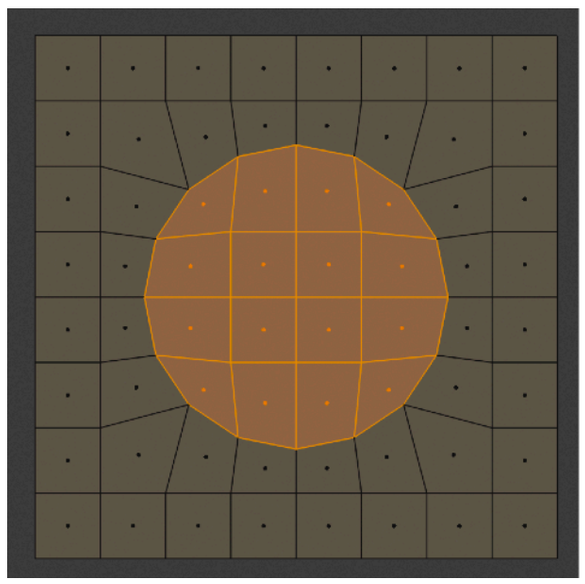
- When there is a **model of the shape** we want to impose to a given geometry, we can iteratively find with a ***non-linear least squares*** method the parameters of the model that best **fit the geometry**
- **For example**, we can find the circumference's parameters (centre and radius) that best fit the set of vertices selected by the user and, then, move those vertices towards the circumference
- Finally, the user can **adjust the obtained results** according to whatever aesthetics criteria (design goals)



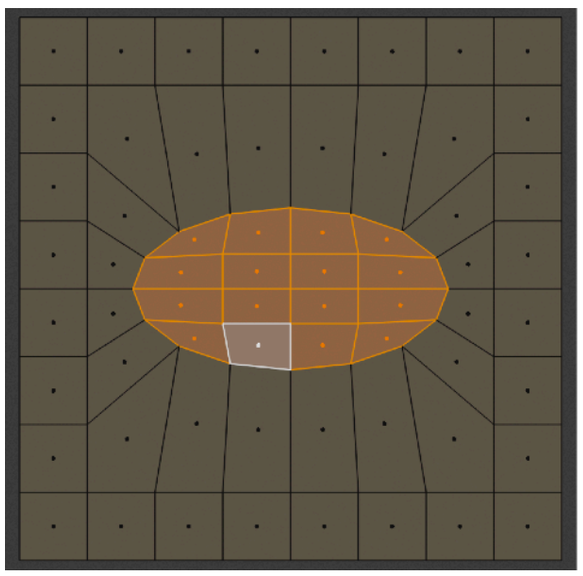
A quad mesh



A few quads selected



The selected quads aligned with the best fit circumference



The fit quads scaled along the vertical axis to meet some design goals

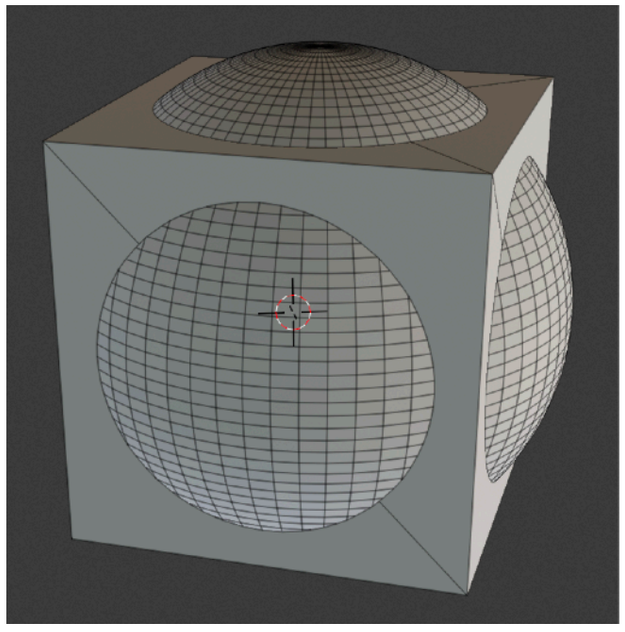
Meshes

Procedural modelling tools - boolean operations

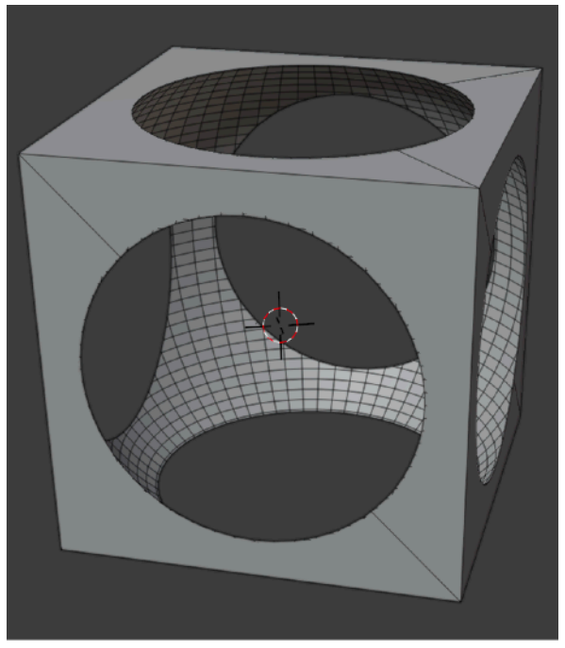
- Complex meshes can be hard to model directly, an issue that can be partially mitigated by using **boolean operations** (*op*) between two meshes (*Mesh_A*, *Mesh_B*) to produce a resulting mesh (Mesh_R): ***Mesh_R*** = ***Mesh_A* op *Mesh_B***
- In practise, it is often assumed that one of the meshes is the one being modified (*Mesh_A*) according to the operation and another *target* mesh (*Mesh_B*): ***Mesh_A* <- *Mesh_A* op *Mesh_B***
- Typical operations (*op*) include: ***intersection***, ***union***, and ***difference***

Results of applying boolean operations

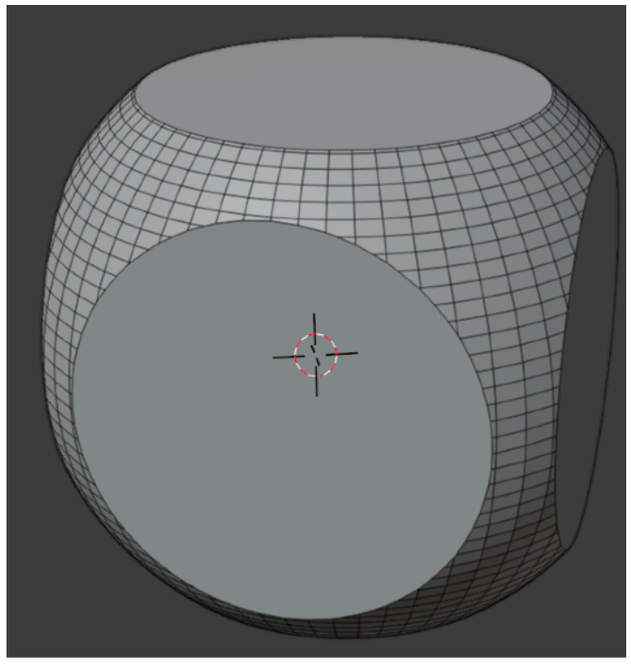
Mesh_A: cube
Mesh_B: sphere



Union: the outer faces of the aggregate volume of Mesh_A and Mesh_B



Difference: the faces of the volume that results from subtracting the volume of Mesh_B from the volume of Mesh_A

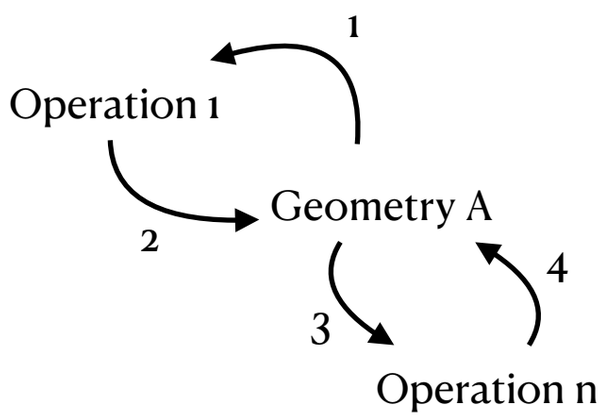


Intersection: the faces of the volume that results from intersecting the volumes of Mesh_A and Mesh_B

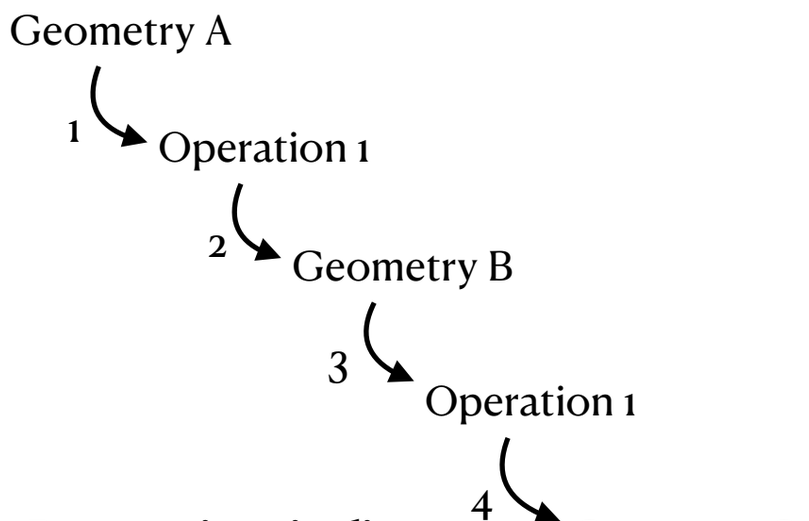
Non-destructive pipeline

Motivation and principles - I

- All operations we have seen so far can be applied in **two distinct pipelines**:
  - **Destructive:** each operation receives a reference to the object being manipulated and alters its geometry, meaning that the object's original geometry is lost forever (similar to multiple processes handling a global variable)
  - **Non-destructive:** each operation receives an object, creates an internal copy, changes it, and outputs the changed object to the next operation, meaning that the original mesh is never altered (similar to passing values as arguments in functions)
    - The non-destructive pipeline can be implemented as an ordered set of operations that perform one after the other without altering the original object, and whose final output is used for rendering purposes



Destructive pipeline

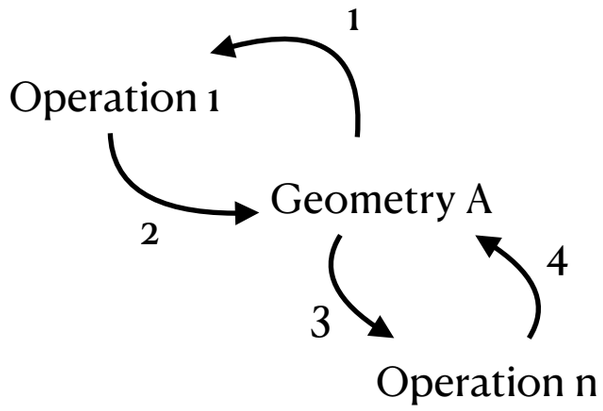


Non-Destructive pipeline

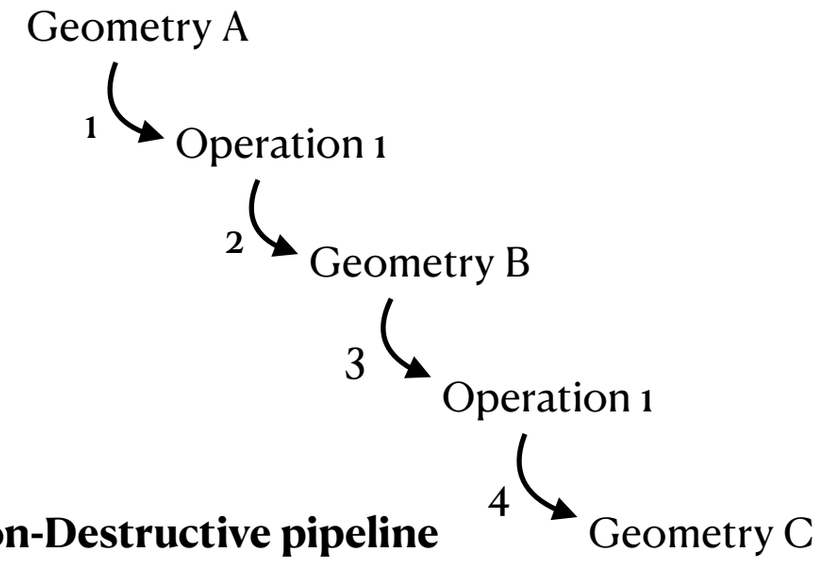
Non-destructive pipeline

Motivation and principles - II

- As the designer explores the design space, one **may need to rapidly change**, at any time, the parameters of an intermediate operation or to add or remove another operation, which can be easily attained in the **non-destructive pipeline**
- The versatility of the non-destructive pipeline comes with a price: it is more **computationally and memory demanding** than the destructive pipeline and the **user cannot directly manipulate** the final geometry's vertices for fine tuning purposes
- In the non-destructive pipeline, **to be able to manipulate** the final geometry's vertices, the user must first **apply all non-destructive operations** to the original mesh and then **switch to a destructive pipeline**



Destructive pipeline

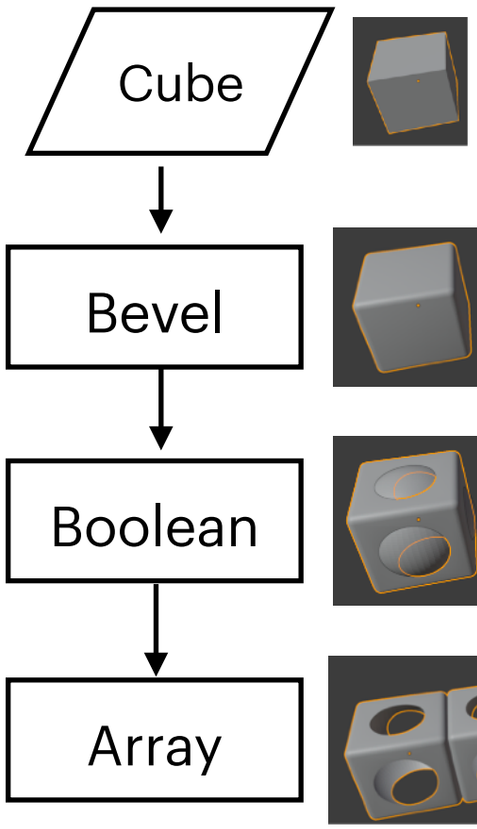


Non-Destructive pipeline

Non-destructive pipeline

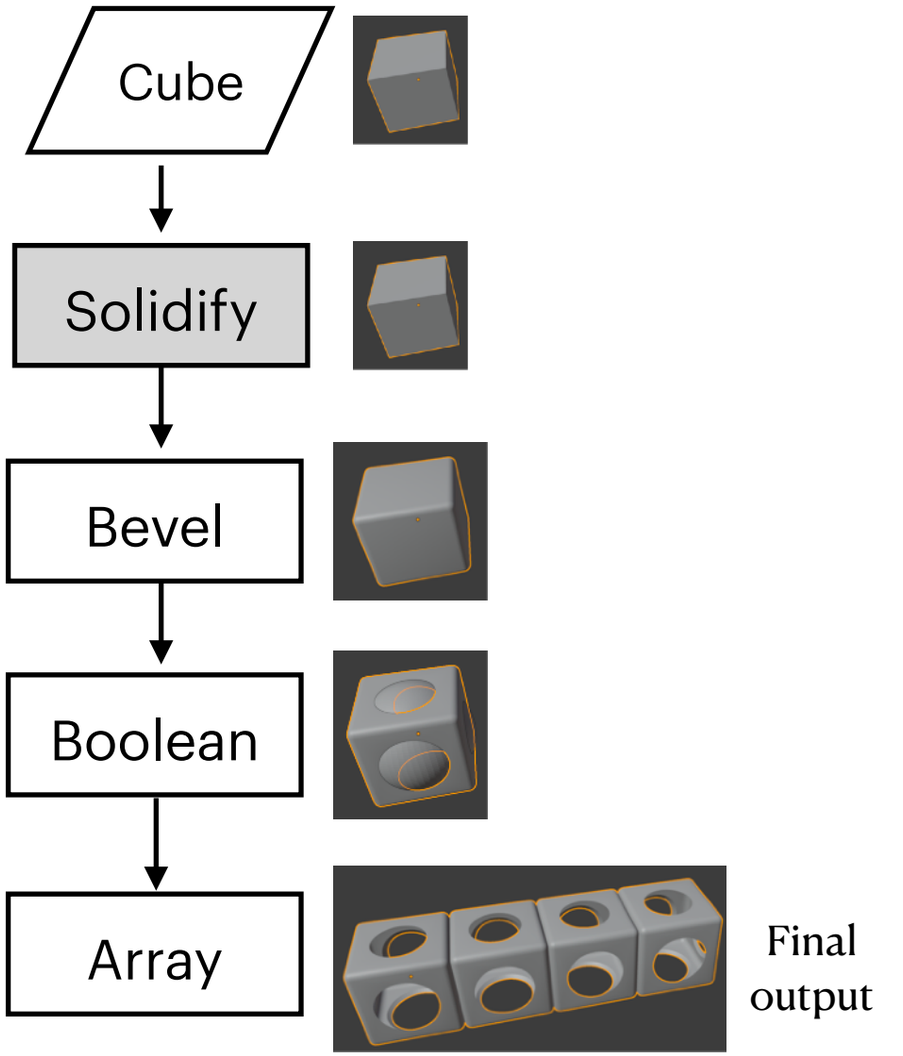
Example - I

1. Imagine we define an ordered set of non-destructive operations to be applied to an initial mesh: a cube



2. Then we realise that we would like to ensure that the object has thickness

3. To do so, we introduce a solidification operation to the ordered set, which results in an immediate change in the final output

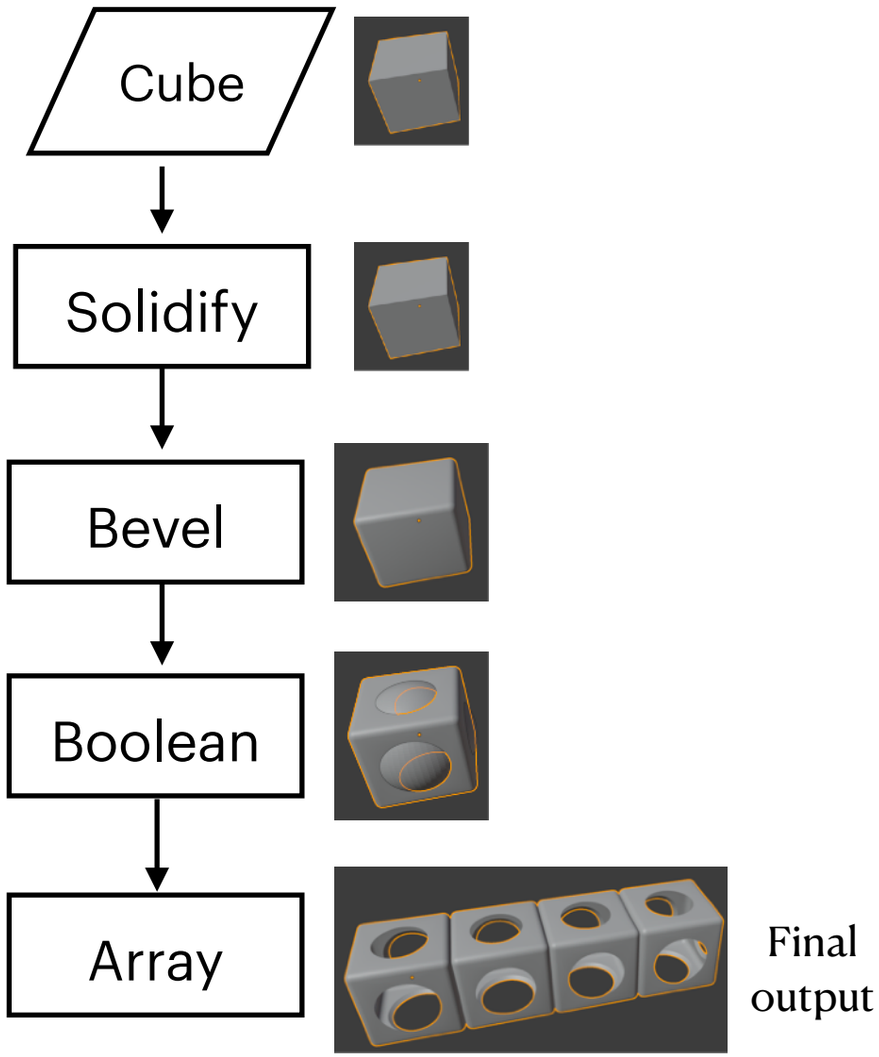


Bevel operation: smoothing hard edges

Non-destructive pipeline

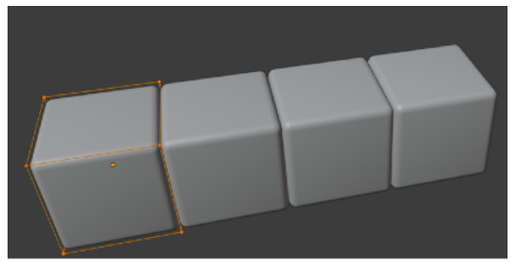
Example - II

4. Now we find that the top of the object should have a different shape

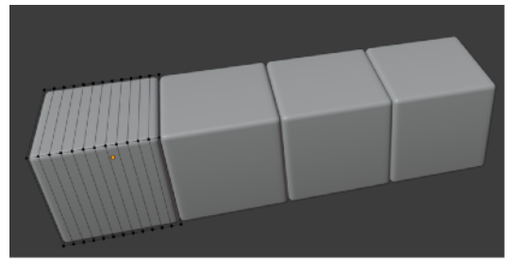


Final output

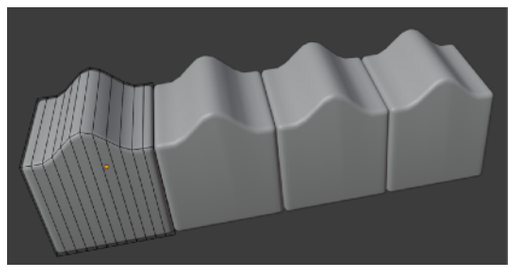
5. To do so, we change the original shape of the cube



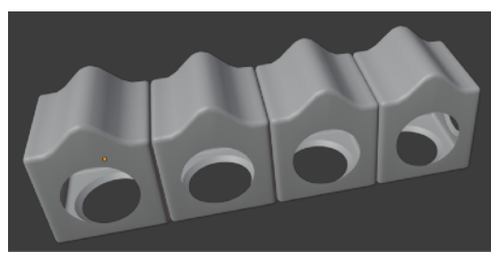
6. By first adding loop cuts (additional detail)



7. And then by performing a vertical translation of an edge with proportional editing



8. Instantly, the output adjusts to our changes

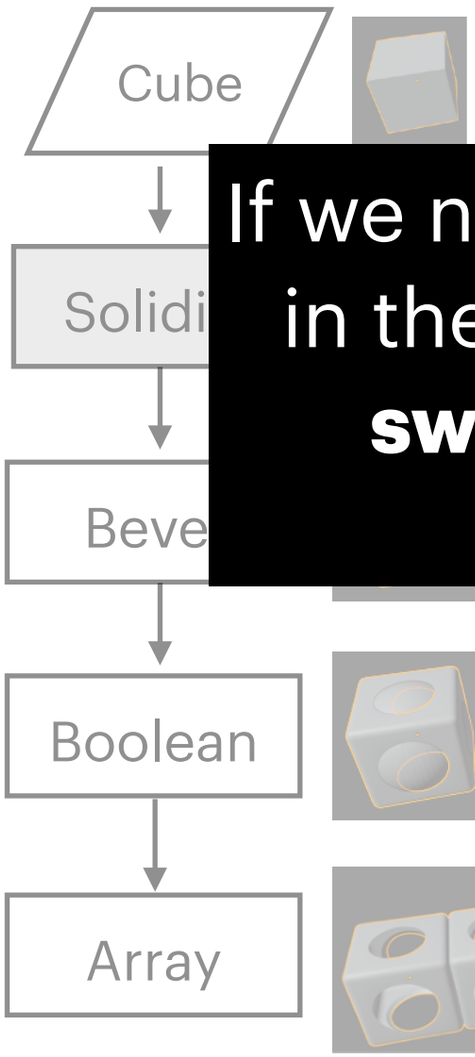


Without the non-destructive pipeline, this task would be error prone and extremely time consuming

Non-destructive pipeline

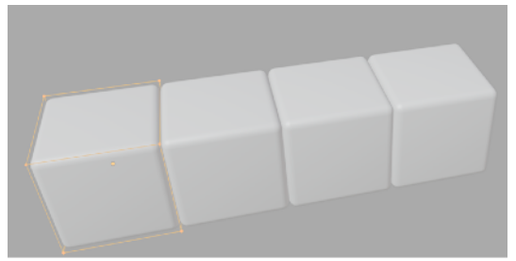
Example - II

4. Now we find that the top of the object should have a different shape

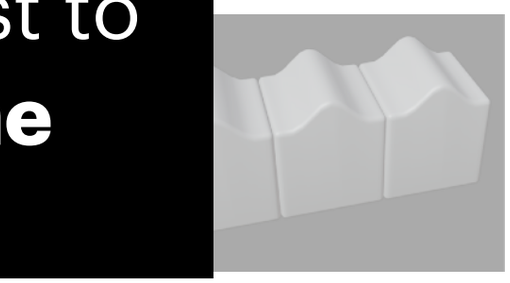


Final output

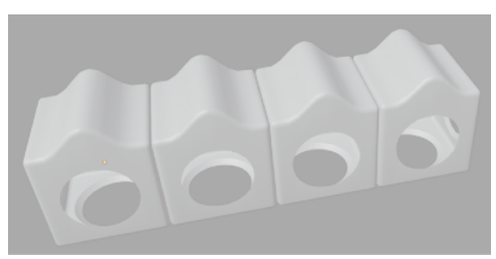
5. To do so, we change the original shape of the cube



6. By first adding loop cuts



8. Instantly, the output adjusts to our changes



If we now want to **fine tune the vertices** in the final geometry, we need first to **switch to a destructive pipeline** (an irreversible step)

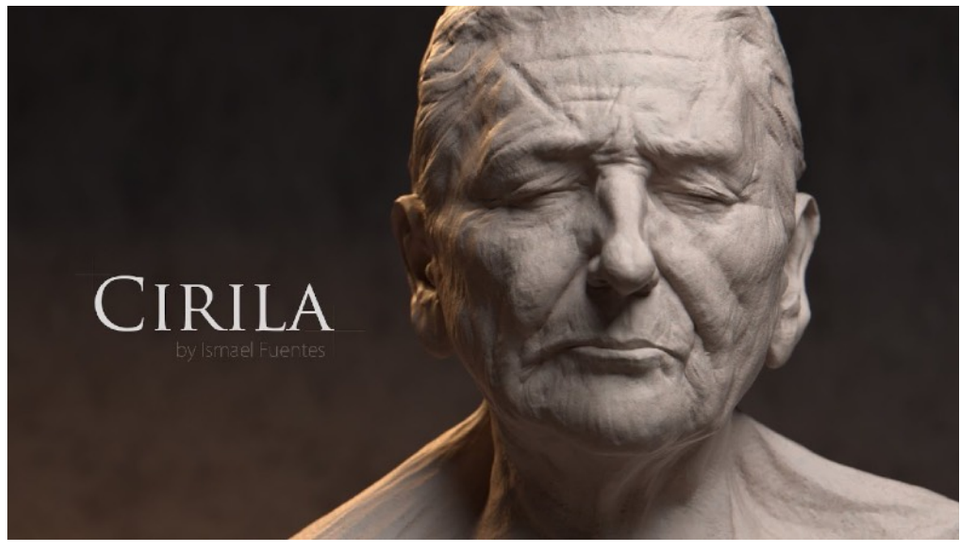
Without the non-destructive pipeline, this task would be error prone and extremely time consuming

Meshes

Clay Sculpting Metaphor

- The use of the ***clay sculpting*** **metaphor** is an alternative to direct direct mesh manipulations, such as extrusions and loop cuts
- This metaphor allows the modeller to create 3D models in a **more intuitive** and **time efficient** way, when compared to direct direct mesh manipulations
- However, the *clay* metaphor depends greatly on the presence of **highly dense meshes** and, thus, with a **significant computational footprint**
- To mitigate this issue, the *clay* metaphor is often employed to **refine a rough 3D model** previously generated with direct mesh manipulations, thus ensuring that **mesh density is only increased where required**

Example of mesh sculpting



[https://www.youtube.com/watch?v=TLb_CCoWn5o]

Meshes

Texture mapping - basics - I

- Once we have a 3D mesh, we need to **paint its surfaces** in the most **efficient** and **realistic** way possible
- The most common approach in computer graphics is known as ***texture mapping***, in which a **2D image** is **mapped** to the **3D geometry**, that is, the image ***wraps*** around the 3D mesh
- This technique allows the designer to **draw images or capture them** with real cameras, setup the mapping between **2D** coordinates in the image and **3D** coordinates in the mesh and use these to colourise the object
- Images can also store different information that can be mapped onto the geometry, such as **reflectance** properties (e.g., where to render highlights in the mesh), **normal** (e.g., where render the illusion of small bumps in the mesh) and **displacement** vectors (e.g., where to actually include 3D bumps in the mesh)

[https://en.wikipedia.org/wiki/Texture_mopping]

This simple animation shows how a single 2D image (an atlas of small texture patches) can be wrapped around a complex 3D object (see the animation it in the associated link)

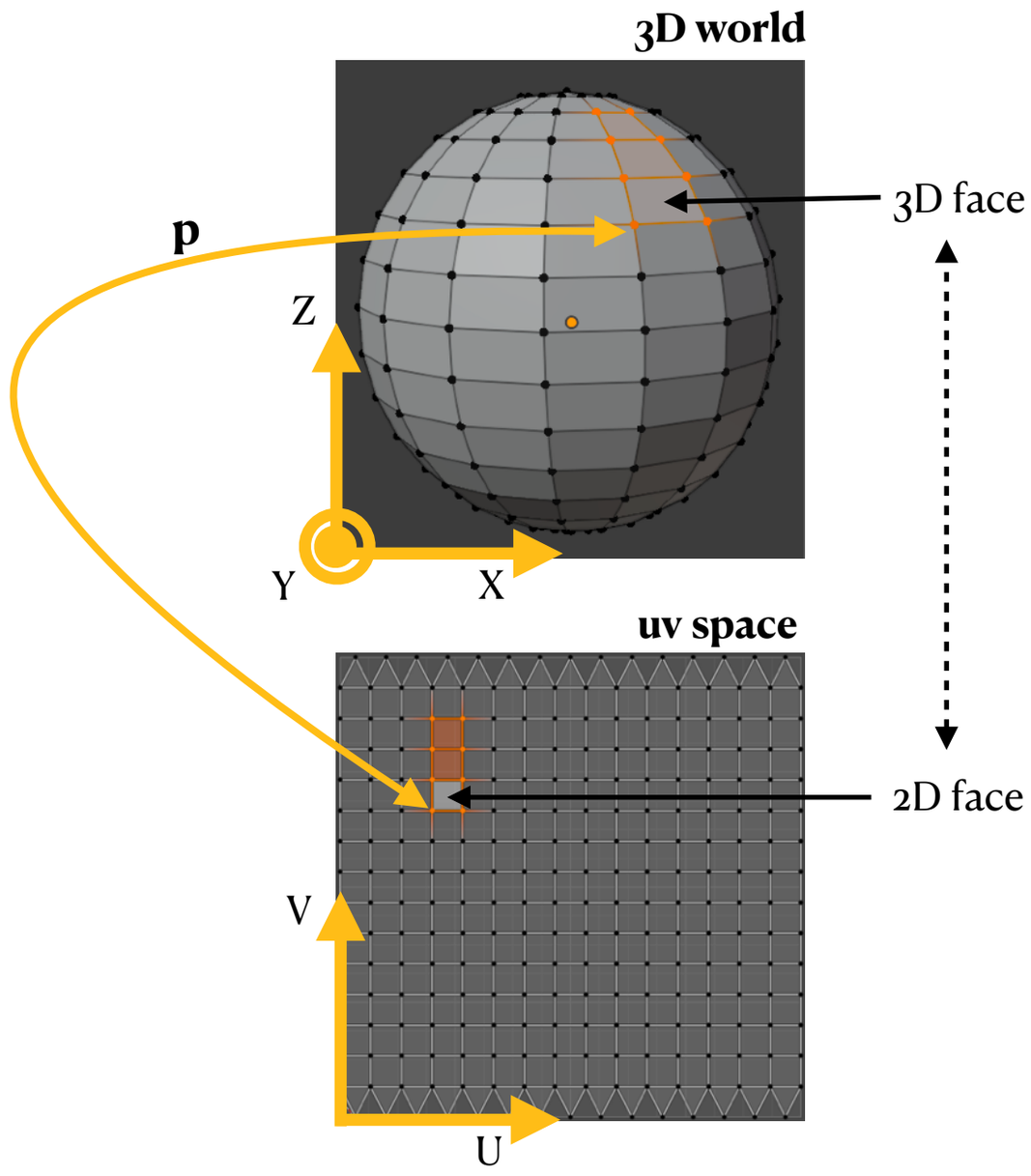
Meshes

Texture mapping - basics - II

- To perform texture mapping, a vertex in our mesh is **5D**, as it includes its XYZ (3D) coordinates in the world and its UV (2D) coordinates in the texture image

A vertex **p**: (X=x, Y=y, Z=z, U=u, V=v)

- Hence, for each 3D face in the mesh, there is a corresponding 2D face in the texture image (**uv space**)



Meshes

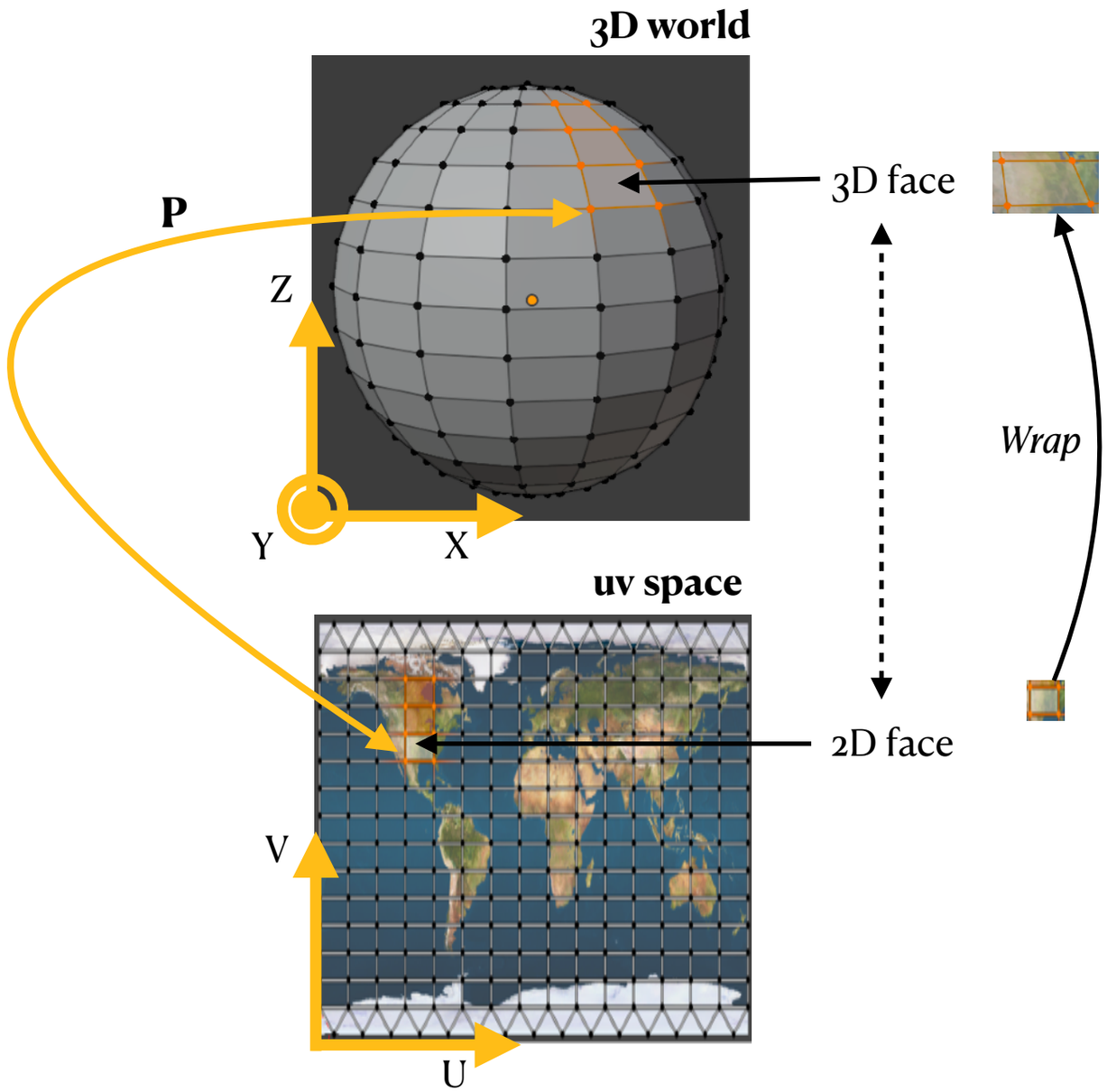
Texture mapping - basics - III

- To perform texture mapping, a vertex in our mesh is **5D**, as it includes its XYZ (3D) coordinates in the world and its UV (2D) coordinates in the texture image

A vertex **p**: (X=x, Y=y, Z=z, U=u, V=v)

- Hence, for each 3D face in the mesh, there is a corresponding 2D face in the texture image (**uv space**)

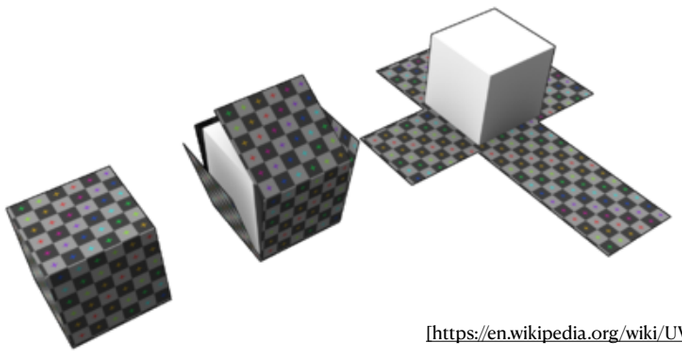
- **Texture mapping**: the image patch covered by a 2D face is *wrapped* in the corresponding 3D face, painting it



Meshes

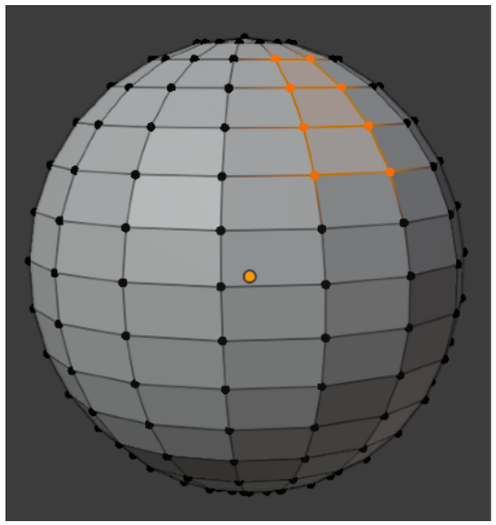
Texture mapping - unwrapping primitives

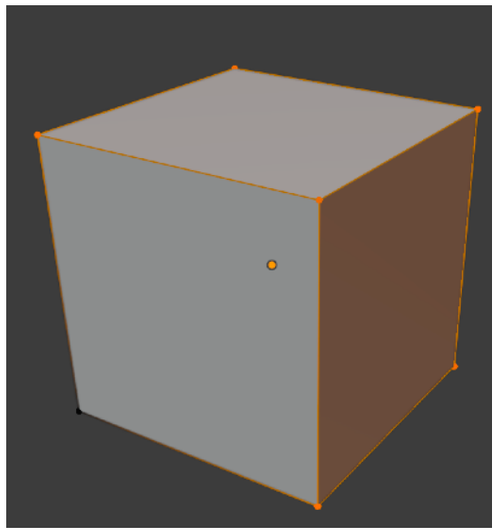
- But, how to ***unwrap*** each 3D face that composes our mesh onto a corresponding 2D face in the uv space? That is, how to unwrap the surface of our object onto a planar surface?
- This is similar to what we do with cardboard boxes:

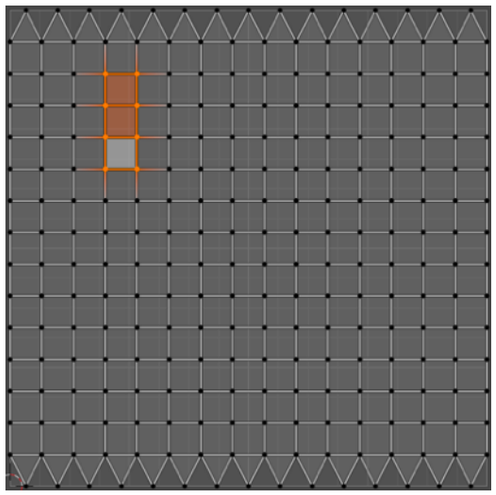


[https://en.wikipedia.org/wiki/UV_mapping]

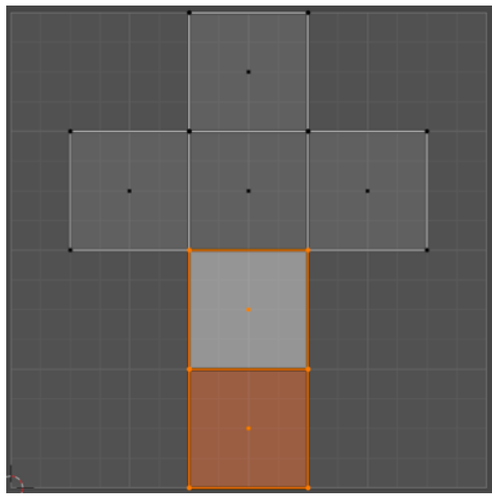
- There are **mathematically** defined **unwrapping** procedures for **primitive shapes**, such as spheres and cubes
- The figures depict the result of applying these procedures







Unwrapping a sphere

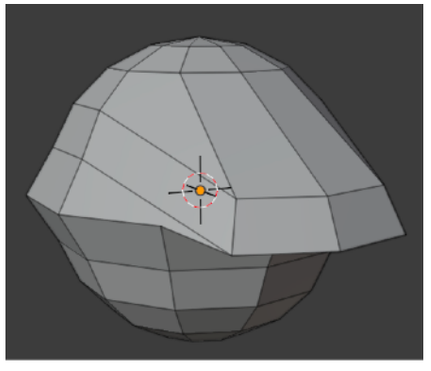


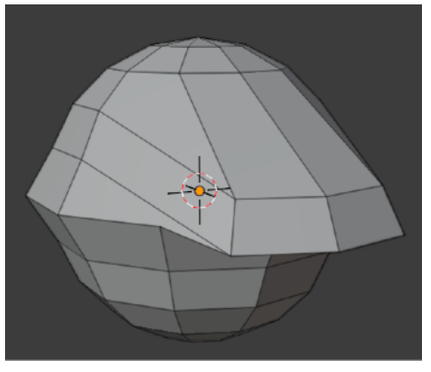
Unwrapping a cube

Meshes

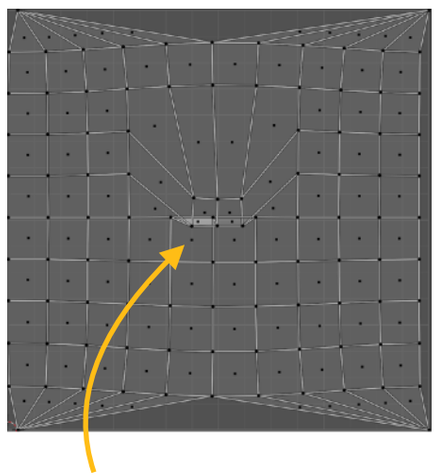
Texture mapping - unwrapping primitives - issues

- Therefore, **if** our mesh resembles a ***primitive shape***, we can **project** our mesh's faces onto it and apply its associated mathematically defined **unwrapping** procedure to *unwrap* the 3D faces
- The less the mesh resembles the *primitive shape*, the more **distorted** and **disorganised** the resulting 2D faces in the uv space will be, thus requiring their **extensive hand-tuning**
- When the 3D mesh as a whole does no resemble a *primitive shape*, we can try to **break it into parts** and handle them separately (e.g., using *parent-child* relationships)



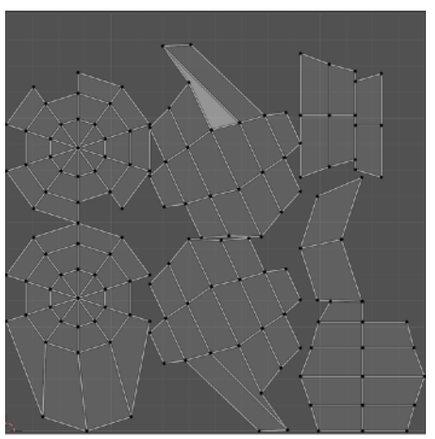


Sphere projection



Overlapping and awkwardly small 2d faces where the "sphere assumption fails the most"

Cube projection



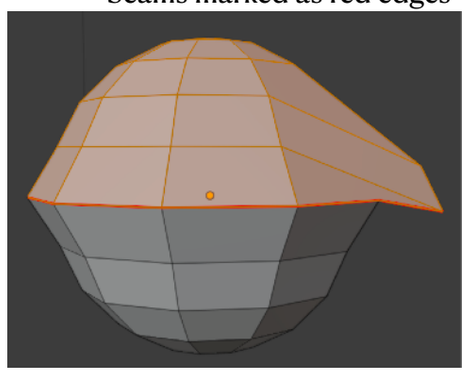
Formation of several "islands" (one per cube face), with discontinuities at their edges, which can cause texture rendering artefacts

Meshes

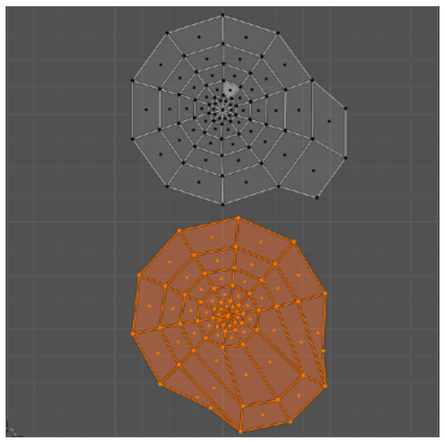
Texture mapping - unwrapping with seams

- **More control** on the unwrapping process can be attained by letting the user defining a set of ***edge seams*** in the mesh and using an algorithm capable of: **breaking** the mesh along those seams; **flattening** the several mesh regions; and **scaling** the resulting 2D faces to bound the texture image
- It is important to ensure that the **seams** are **placed** such that **discontinuities** in the texture mapping process are only present where do **not induce rendering artefacts**
- Seams can segment the mesh into separate parts, generating ***islands*** of 2D faces in the uv space, which can be handy to **move these around** the image texture, but they also introduce more discontinuities
- The **generation of the seams** can be **automatised** by algorithms that inspect, for instance, the presence of **sharp edges** in the 3D mesh

Seams marked as red edges

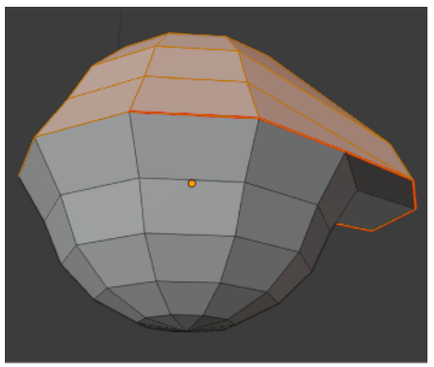


Seams-based unwrapping ↓

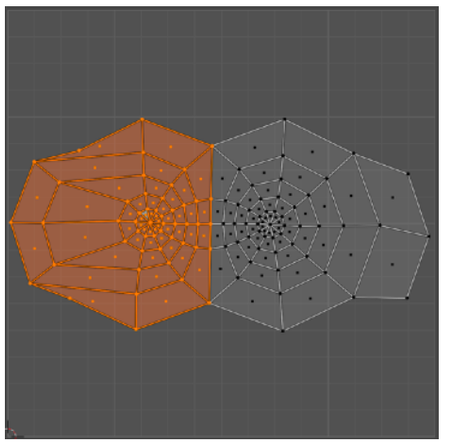


A whole-around seam allows splitting both up bottom, reducing island discontinuity issues to the up-bottom separation

Seams marked as red edges



Seams-based unwrapping

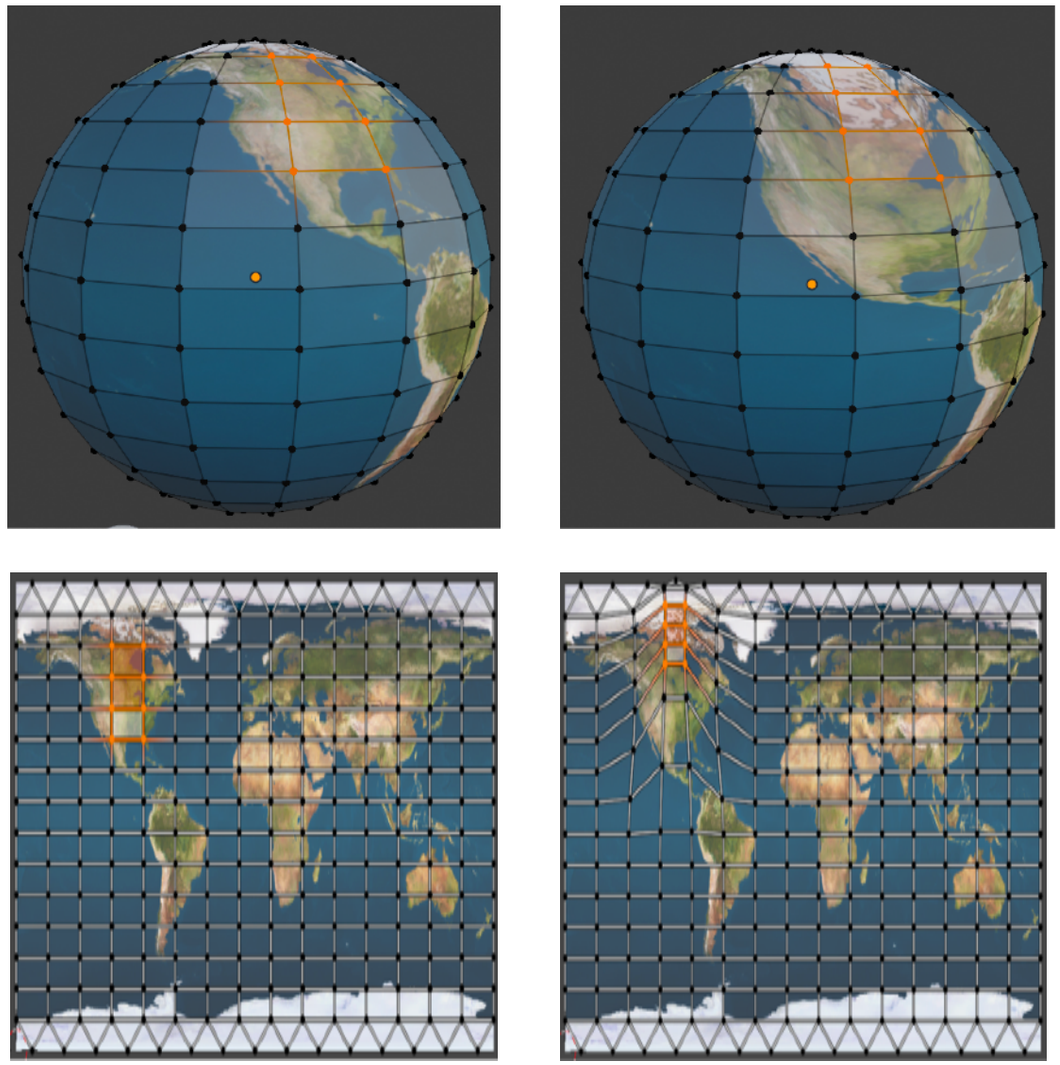


A not-whole-around seam avoids the creation of islands, which can be good (reduces texture discontinuities) or bad (it can be handy to move the top and bottom parts across the texture image)

Meshes

Texture mapping - refining uv mapping - I

- After running the unwrapping algorithms, the designer may **refine the 2D faces** in the uv space, similarly to the way 3D faces are manipulated
- In fact, the user can **manipulate** the **3D** and the **2D** faces, **independently**
- These adjustments can, for instance, be employed for instance to adjust how much **area** of the image is covered by a 2D face, which is important to **avoid distortions** and **blur** in the rendering of the corresponding 3D face
- The figures show the impact of translating several 2D faces: the image patch rendered in the 3D model changes, as well as its quality

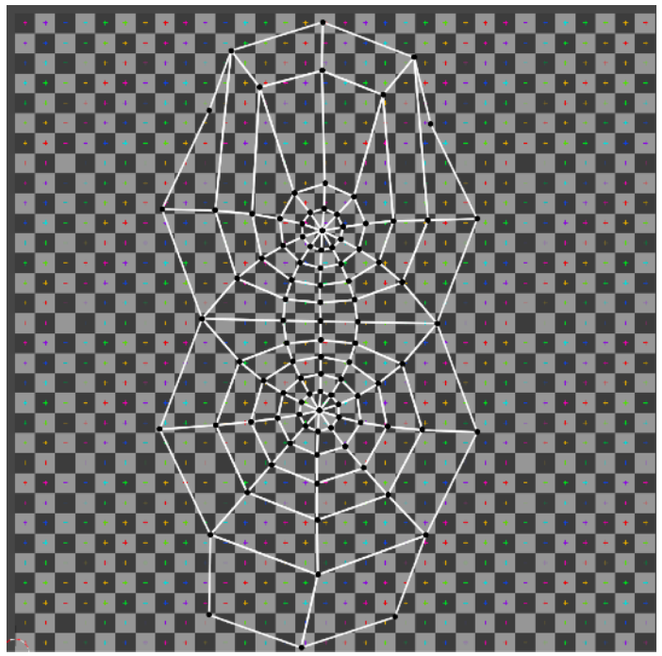


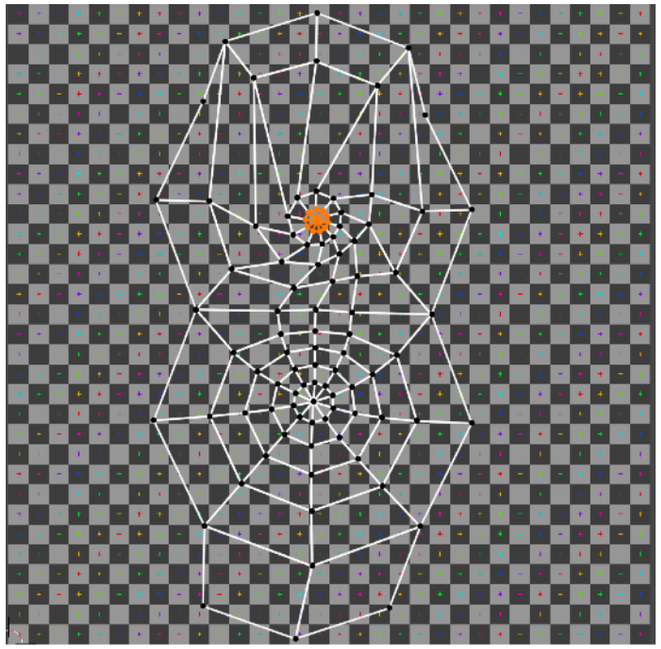
After the transformation, the 3D faces cover fewer pixels in the 2D image texture and, thus, the rendering gets blurred

Meshes

Texture mapping - refining uv mapping - II

Original uv space (Unwrapping using seams)

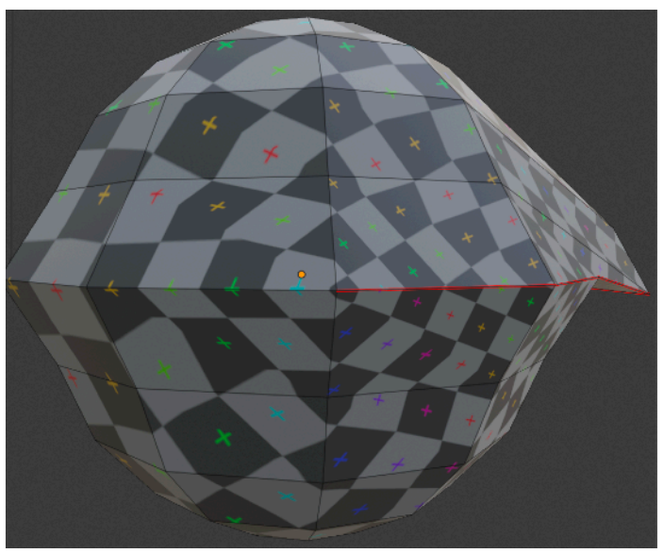


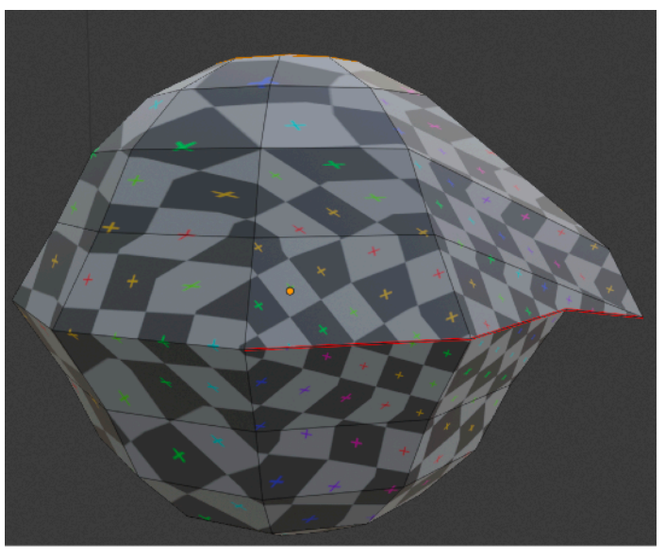


Transformed uv space (Rotation of selected faces with proportional editing)

3D mesh with seams in red

(Note the texture discontinuities across seams)





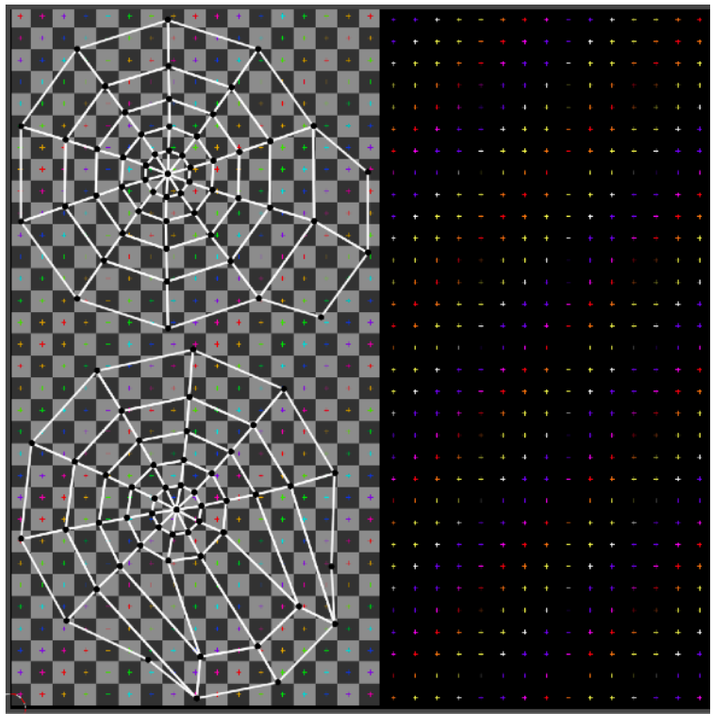
3D mesh with seams in red

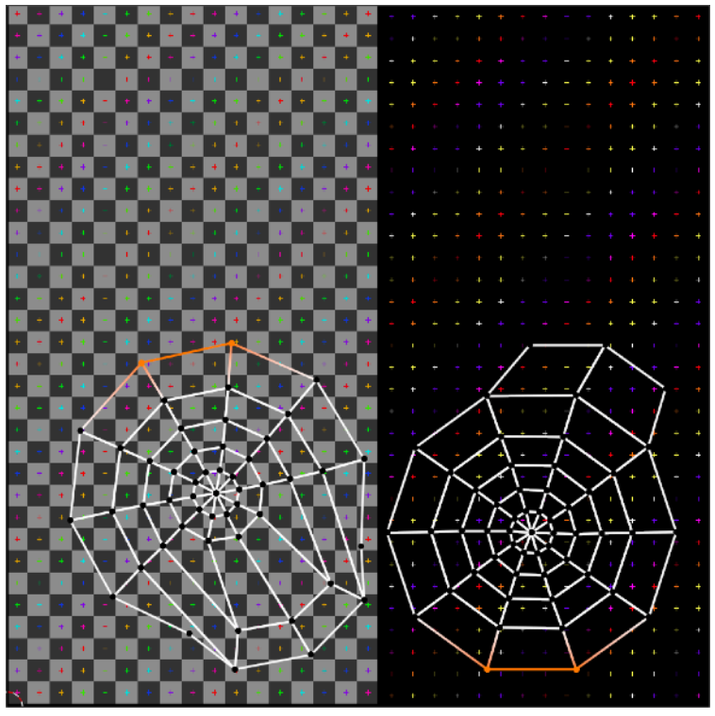
(Note the texture discontinuities across seams)

Meshes

Texture mapping - refining uv mapping - III

Original uv space (Unwrapping using seams)

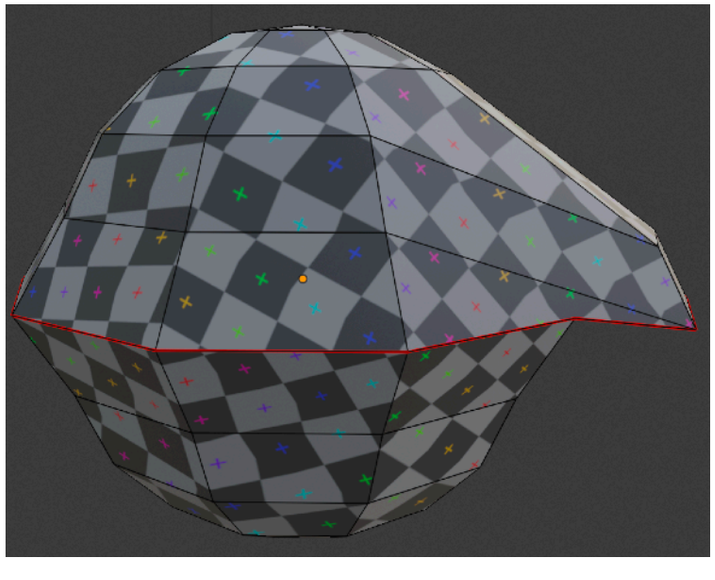


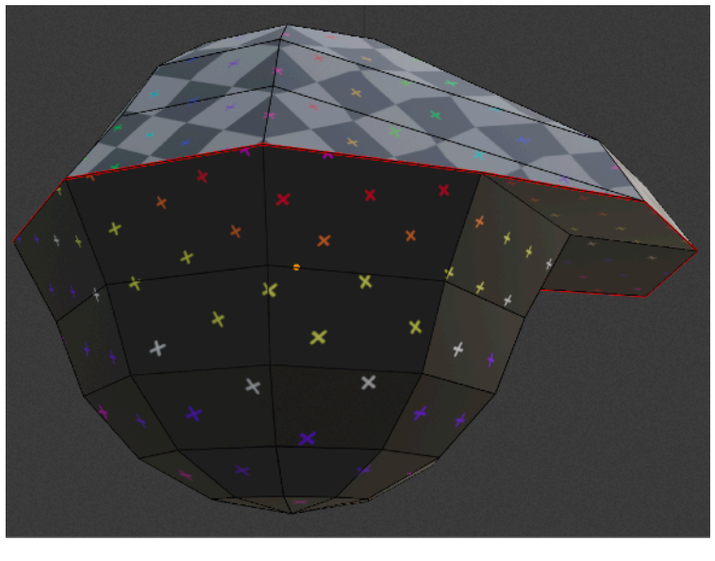


Transformed uv space (Translation of top faces to the right)

3D mesh with seams in red

(Note the texture discontinuities across seams)





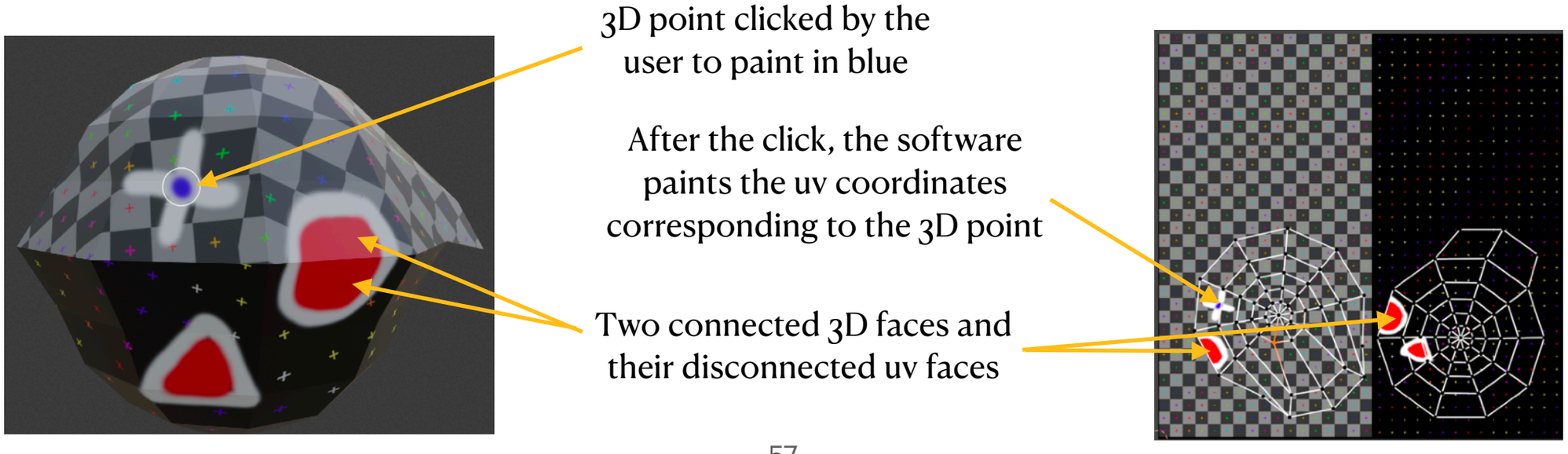
3D mesh with seams in red

(Note the change in terms of the texture mapped)

Meshes

Texture mapping - interactive painting - I

- The texture to be mapped to a given geometry can be generated in many ways:
  - Procedurally: algorithms that generate the colour of each texture's pixel (e.g., *perlin noise*)
  - Using real cameras: camera photographs can be used as texture images, but care must be taken to handle highlights and perspective effects
  - Drawing on texture image: image editing tools (e.g., brushes) can be used to paint the texture image directly
  - Drawing on 3D mesh: brushes can also be used directly over the 3D surface, in which case the software transforms the 3D point under the mouse pointer to its corresponding uv-coordinates, given the unwrapping procedure initially applied to the mesh, and, then, paints the image texture in the obtained uv location
- Drawing on 3D meshes handles easily the situations in which connected 3D faces have disconnected 2D uv faces

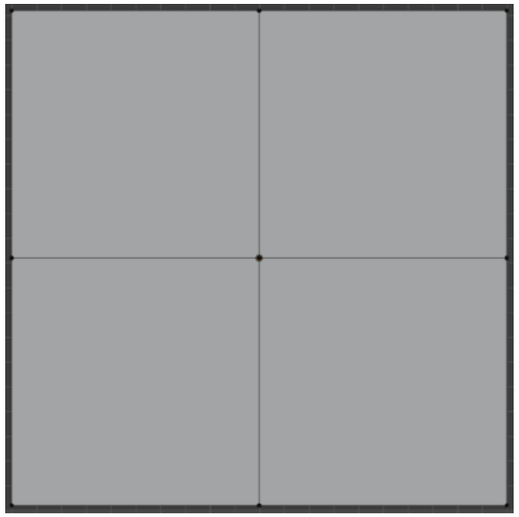


Meshes

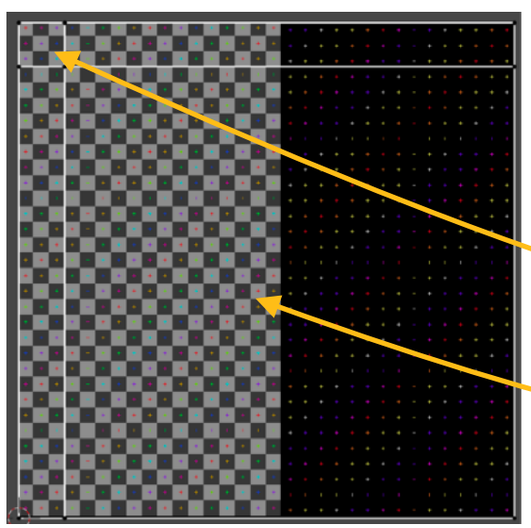
Texture mapping - interactive painting - II

- Bear in mind that the area of the 2D face determines the resolution with which we can paint the corresponding 3D face

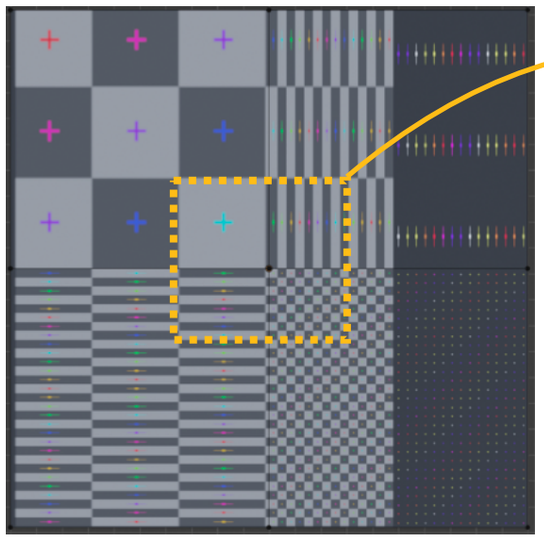
A 3D mesh composed of four quads



The uv faces over the texture image



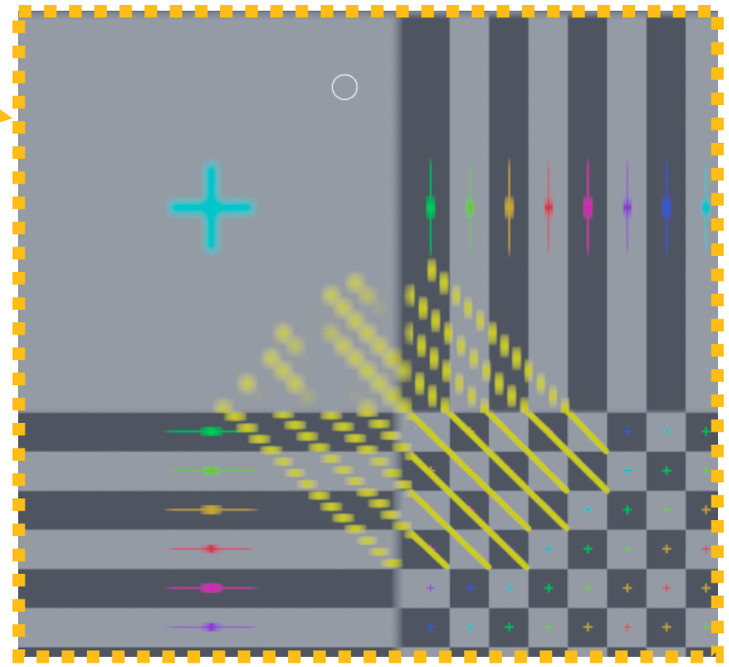
The 3D mesh with the texture mapped



A uv face with small area

A uv face with large area

A patch of the result after painting lines on the 3D mesh directly using the mouse

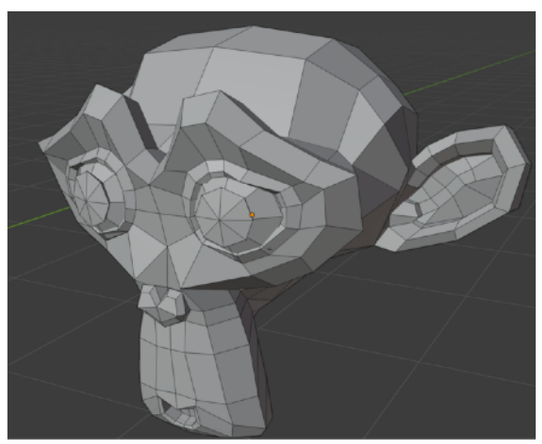


The resolution of the lines changes according to the pixel area under the corresponding uv faces

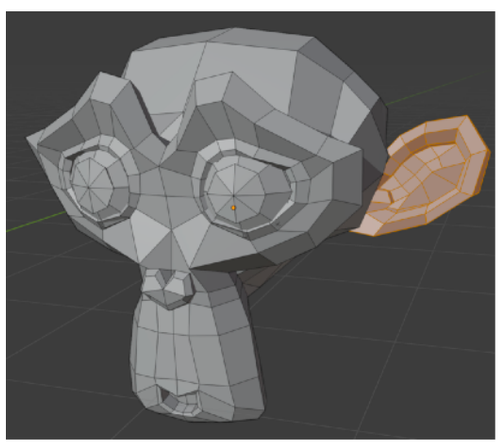
Meshes

Modularity - Vertex Groups

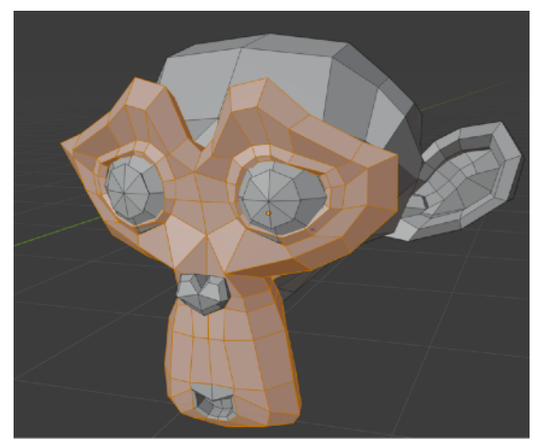
- *Parenting* is a way of hierarchically composing an object using several meshes, allowing their re-use (e.g., a wheel can be replicated in several cars)
- *Parenting* also allows the designer to manipulate each object's part in an isolated way (e.g., applying a transformation with proportional editing)
- However, given object's parts are topologically separated, animation algorithms fail to morph the meshes in an orchestrated manner
- To keep meshes joined for the animation sake and still provide some design modularity, *vertex groups* can be used
- A vertex group is a simple list of references to vertices, which can be assigned and removed from the list by the user
- The elements of a vertex group can then be instantly selected by the user for subsequent edition



A mesh



The faces of a *vertex group* selected

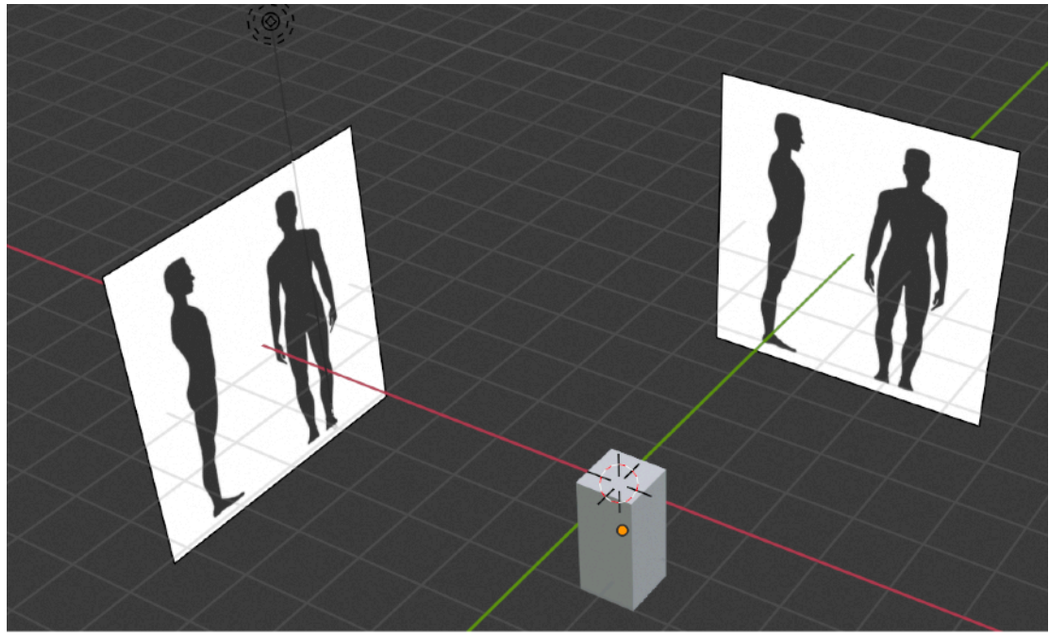


The faces of another *vertex group* selected

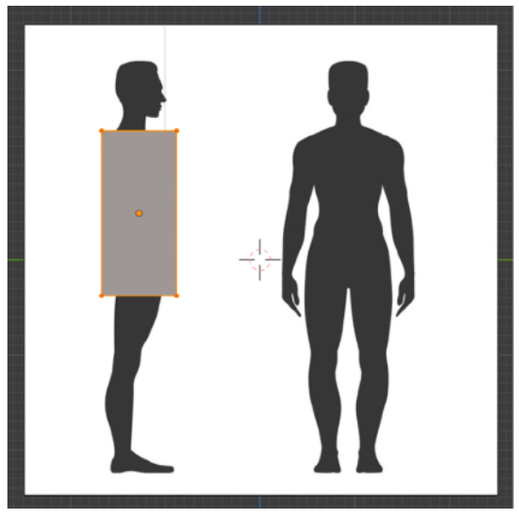
Meshes

Reference images

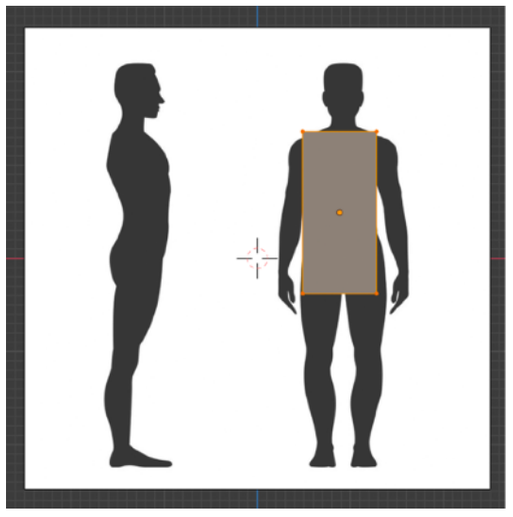
- To help in the 3D modelling, modellers often use **reference 2D images** aligned with the world axes
- These images can be pictures of **real objects** that the modeller is trying to mimic or they can be **hand-drawn sketches** of the target 3D model



A scene with two reference images to help modelling a human from a cube



Lateral orthographic view



Frontal orthographic view