PolyFrame Beta 0.1

What is PolyFrame?

PolyFrame is a structural form finding and analysis plug-in for the popular modeling tool Rhinoceros3d. PolyFrame uses dual and reciprocal polyhedron structures to simultaneously design and control the flow of forces and the optimal form of a structure. The form finding and control mechanism are based on the principles of 3d Graphic Statics.

PolyFrame is based on a polyhedral construction and manipulation environment called PolyFramework.

Data structure

In order to facilitate the manipulation of polyhedral clusters PolyFramework employs a data structure used to maintain the topological and geometrical relationships that define the cluster. Each complete PolyFrame structure (also known as PFoam) contains elements that encode all the subparts of the cluster: cells (PFCell), faces (PFFace), edges (PFEdge) and vertices (PFVertex).

Naming convention

The majority of the PolyFrame routines are based on the relationship between two polyhedral structures one depicting the force and the other representing the form. The two structures are always the dual of one-another and they can also be reciprocal if the edges of one are perpendicular on the faces on another. These two conditions duality and perpendicularity are the essential prerequisites of static equilibrium in the structural design process that PolyFrame makes possible. As a result of this connected relationship between two diagrams, a naming convention has been established to keep things in the dynamic context of form-finding and optimization. A diagram loaded, created from geometry or manipulated will be referred to as **primal**. A diagram derived from or connected the first one will be called **dual**. The primal/dual denominations are irrespective or the form/force designation of the diagrams and they can be interchanged depending on which diagram is the main subject of the command at one time.

PolyFrame and PolyFramework are developed by the Polyhedral Structures Lab at PennDesign, University of Pennsylvania.

Principal Investigators

Dr. Masoud Akbarzadeh, Dr. Andrei Nejur

Software Developers

Dr. Andrei Nejur, Dr. Masoud Akbarzadeh

Command descriptions.

For now, PolyFrame adds 7 new commands to the Rhino environment

PFBuild 

PFBuild creates a PolyFrame structure using native Rhino brep geometry.

- The accepted **input** for the command can consist of polyhedrons represented either as individual faces or as polysurfaces. The command has three options.
- **PointColapseLimit** governs the minimum distance between two breps vertices required to be considered separate. Under this limit all vertices collapse into one single one. This is useful when dealing with non-perfect input where source geometry has tiny overlaps or gaps. Setting this too low can cause geometry to remain disjoint thus rendering the input unusable or producing undesirable result. Setting PointColapseLimit to high (in general higher that the shortest edge in the input) can cause edges to collapse into single points with all the consequences that can derive from that.
- **PlanarityMaxDeviation** sets the maximum accepted perpendicular deviation of vertices from the average plane of the input faces. While PolyFrame can process non planar breps as input, large deformations can cause unexpected errors in the cell finding process. A value to small can cause the command to reject some of the otherwise planar input as non-panar.
- **ReplaceGeo** decides whether the constructed PolyFrame will replace the input geometry or it will be placed somewhere else in the document.
- **Output options**
    - **Type** selects the type of container geometry for the PolyFrame data structure. The possible values are Edges – for line based geometry; BrepFaces for trimmed surfaces representing each face; BrepCells for polysurfaces representing each cell; MeshFaces for individual meshes for each face; MeshCells for a mesh based representation of each PolyFrame cell.

PFDual 

PFDual creates the dual of a PolyFrame structure. Each element in the input (the primal) has a counterpart in the dual. Primal Cells -> Dual Vertices ; Primal Faces -> Dual Edges ; Primal Edges -> Dual Faces; Primal Vertices -> Dual Cells.

- **Input** for the command is a PolyFrame container from the Rhino document. Several options are available for the input stage.

- **UpdateGeo** decides whether the input PolyFrame will be updated with the geometrical manipulations that occurred since it was last saved into the document.
- **ClearConstraints** specifies whether the geometry constraints present in the PolyFrame data will be cleared upon loading. This setting has no immediate consequences on the command results.

- **Output options**
  - **Type** [Edges, BrepFace, BrepCells, MeshFaces, MeshCells]

PFPerp

PFPerp establishes the reciprocity relation between a primal and its dual. Using the topological relationship, the tool iteratively moves the vertices of the primal until the primal edges are perpendicular to the dual face planes. The command has layers of options for input, processing, constraints and output.

- **Input**
  - **UpdateGeo** assumes or discards the transformations of the container geometry
  - **ClearConstraints** allows the user to decide whether the saved transformation constraints present in the PolyFrame data will be maintained or not. See below for clarifications regarding constraints
- **Processing**
  - **MaxIterations** controls the maximum number of cycles the tool is allowed to make before it will stop. 1000 is a conservative value, especially for more complex models consider increasing it. Even if the user sets this value too high the iteration will stop if the solution reaches prescribed maximum deviation (see next option), solution stagnates (no significant vertex movement is recorded between consecutive steps), or user pressed <ESC> key.
  - **MaxDeviation** stipulates the maximum allowed angle deviation between a primal edge and a dual face normal. If all angles between primal edges and dual face normal are under this value, the iteration stops.
  - **MinEdgeLength** defines the minimum length a primal edge is allowed to shrink to during the process. Please note that having a large value here will cause the structure to grow and depending on the configuration can be detrimental to the fast convergence towards a small maximum deviation. On the other hand, setting the value to small can produce very short edges or can cause some edges to flip during the process. Best value is usually determined through experimentation for each case.
  - **SetEdgeLength** allows the user to set target lengths for the edges in the primal. This is a perpedincularization constraint. The option opens a new dialogue that allows for the selection of edges in the PolyFrame. The user can select one edge at a time and set its target length in a dedicated pop-up dialogue. If multiple

edges are selected in a sequence the provided target length will apply to all of them.

- **SetVertPos** allows the user to set position in the primal. This is a perpedincularization constraint. The option opens a new dialogue that allows for the selection of vertices and subsequent position constraints for them. The user can select either one vertex or multiple vertices in sequence. Once the selection set is confirmed another layer of options is presented to the user. By default, the user can click on any geometry in the document and use it as a constraint. There are two options in the command line at this time. **Outside** has meaning only for closed constrain geometry and specifies whether the point will be kept inside the geometry rather than on its surface. **Fixed** simply constrains the selected vertices to their present positon.
- **UpdateDualGeo** like the name implies can update the dual geometrical information stored in the primal based on a topologically equivalent dual PolyFrame container present in the rhino document. For this operation to succeed the new dual needs to be related to the primal being perpendicularized.

- **Output options**
  - **Type** [Edges, BrepFace, BrepCells, MeshFaces, MeshCells]

PFPlanarize 

PFPlanarize as the name implies, attempts to make planar all the faces of a PolyFrame or cluster of trimmed surfaces/polysurfaces. The process moves vertices of the input geometry that are out of their respective face planes until all faces are planar within a provided tolerance. Like PFPerp the command allows the user to interact with options on multiple layers.

- **Input**
  - **UpdateGeo** assumes or discards the transformations of the container geometry
  - **ClearConstraints** allows the user to decide whether the saved transformation constraints present in the PolyFrame data will be maintained or not. See below for clarifications regarding constraints
- **Processing**
  - **Maximum allowed vertex deviation** stipulates the maximum allowed distance between a vertex and the average plane of a face containing that vertex. If all vertex distances from face planes are under this value, the iteration stops. In Rhino surface with vertex deviations less than 1e-4 (0.0001) units are considered planar. PolyFrame can work with much larger deviations from planarity. The only problem may arise if BrepFaces or BrepCells are chosen for the container geometry and faces are not planar. The geometry will still be

created but may have small imperfections and take longer to be visualized. The default set value should work for most cases.

- **MaxIterations** controls the maximum number of cycles the tool is allowed to make before it will stop. 5000 is a conservative value, especially for more complex models consider increasing it. Even if the user sets this value too high the iteration will stop if the solution reaches prescribed maximum deviation solution stagnates (no significant vertex movement is recorded between consecutive steps), or user pressed <ESC> key.
- **Algorithm** allows the user to choose between two planarization algorithms. **SType** or soft planarization allows for the use of constraints but is a little slower and less capable to reach really small deviations because it also checks for stagnating solutions during the process. **HType** or hard planarization ignores any constraints and has no other automatic stopping criteria than MaxIterations or Maximum allowed deviation but is faster and more capable to reach really small deviations. HType is usually recommended if no geometry constraints need to be imposed.
- **MinEdgeLength** defines the minimum length an edge is allowed to shrink to during the process. Leaving this to the default value is usually a good idea as the only concern here is edge flipping during the planarization of extreme deviations from plane.
- **SetVertPos** allows the user to set position in the primal. This is a constraint. The option opens a new dialogue that allows for the selection of vertices and subsequent position constraints for them. The user can select either one vertex or multiple vertices in sequence. Once the selection set is confirmed another layer of options is presented to the user. By default, the user can click on any geometry in the document and use it as a constraint. There are two options in the command line at this time. **Outside** has meaning only for closed constrain geometry and specifies whether the point will be kept inside the geometry rather than on its surface. **Fixed** simply constrains the selected vertices to their present positon.
- **SetEdgeLength** allows the user to set target lengths for the edges in the primal. This is a constraint. The option opens a new dialogue that allows for the selection of edges in the PolyFrame. The user can select one edge at a time and set its target length in a dedicated pop-up dialogue. If multiple edges are selected in a sequence the provided target length will apply to all of them.
- **SetFaceArea** allows the user to set target areas for faces in the system. The option opens a new dialogue that allows for the individual selection of faces and subsequent specification of target areas. During the planarization process faces will be scaled to meet the target areas

- **Output options**
  - **Type** [Edges, BrepFace, BrepCells, MeshFaces, MeshCells]

PFSwitch 

PFSwitch allows the user to switch the container representation of any PolyFrame. The command simply loads the data from the document with the usual options and saves it again replacing or duplicating the original while allowing the use to pick another type of geometric representation.

- **Input**
    - **UpdateGeo** assumes or discards the transformations of the container geometry
    - **ClearConstraints** allows the user to decide whether the saved transformation constraints present in the PolyFrame data will be maintained or not.
- **Output options**
    - **Type** [Edges, BrepFace, BrepCells, MeshFaces, MeshCells]

PFPipe 

PFPipe creates a volumetric representation of a PolyFrame based on its container and type. If the selected PolyFrame represents a form diagram, the volumetric representation will use the force magnitude from its dual (the area of the dual faces) to create cylinders around its edges. If the selected PolyFrame represents a force diagram the command will extract its dual from the data structure and create the piped representation based on it. The produced pipes will have radii and colors according the magnitude of the forces in the dual.

- **Input**
    - **UpdateGeo** assumes or discards the transformations of the container geometry
    - **ClearConstraints** allows the user to decide whether the saved transformation constraints present in the PolyFrame data will be maintained or not.
- **Processing**
    - **MinRadius / MaxRadius** control the interval the radii of the pipes. The force magnitude values read from the dual will be “normalized” between those two values while keeping the relative ratio between them.

PFTransform 

PFTransform allows for the transformation of a PolyFrame structure while keeping the direction of its face normals constant.

- **Input**

- **UpdateGeo** assumes or discards the transformations of the container geometry
- **ClearConstraints** allows the user to decide whether the saved transformation constraints present in the PolyFrame data will be maintained or not

- **Processing.** The command allows for individual vertices to be picked. The vertices can then be moved either freely or along connecting edges. The algorithm propagates the changes in the whole structure following the rules of parallelism. The command shows a live preview of the transformation color-coding the edges with either blue if they remain with the same direction or red if they flip.

PolyFrame Workflows

Simple perped structure

The simplest typical workflow using PolyFrame is made of just three steps. It implies creating a readily optimized structural form from a set of polyhedrons representing the forces in the structure.

Prerequisites.

In order to begin a start geometry representing the force is needed. This must be present in the Rhino document as an aggregation of trimmed surfaces representing the faces of the force polyhedrons. It is important that the faces are built and assembled together in the document without significant discontinuities or overlaps and that they are as planar as possible. In the event that the input geometry does not meet the minimum criteria to be turned into a PolyFrame the tool will abort and bake the problem geometry into a separate layer entitled <<Error Geometry>> in orange.

Step.1

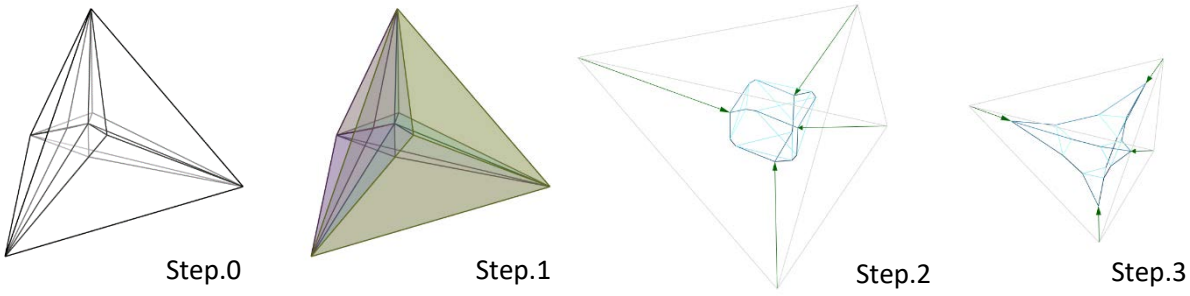
Using PFBuild a PolyFrame is constructed from the input geometry. The default representation of the PolyFrame will be edges. Also by default the original input geometry will be replaced by the PolyFrame container. The image in step one shows a BrepCell representation of the constructed PolyFrame for to better distinguish it from the input geometry

Step.2

Create the dual of the built PolyFrame (the primal) using PFDual and the default options.

Step.3

Perpendicularize the new structure using the default options.



Step.0 Step.1 Step.2 Step.3

Perping a PolyFrame with constraints

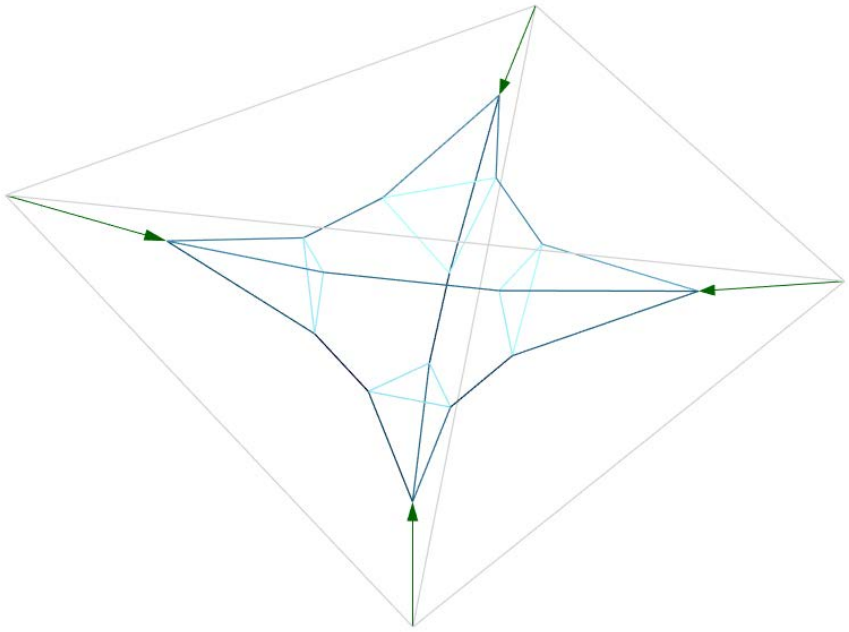
This workflow will detail how to perpendicularize a PolyFrame structure with added constraints like certain positions for the vertices or certain set lengths for the edges.

Prerequisites

In order to start this workflow a form PolyFrame (the dual of a force) is required. Although reciprocity is not required as a starting condition it is advisable that the PolyFrame being constrained through perping starts from an equilibrium (reciprocal to form) state. We can actually use the last step from the previous workflow.

Step.1

Load the form PolyFrame from the PFPerp command

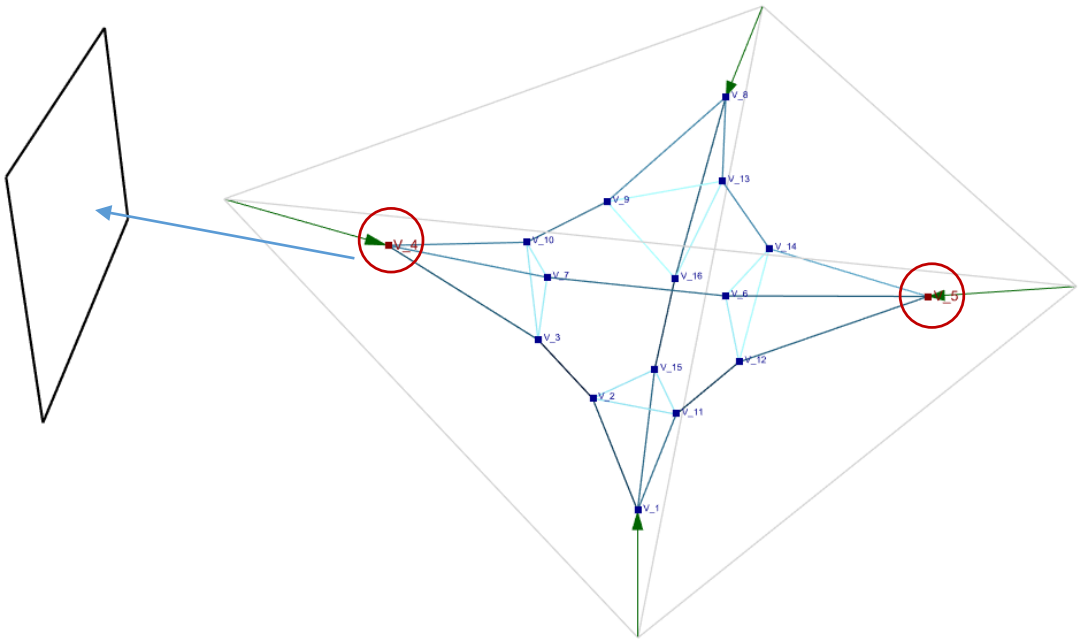


Step.2

Set one vertex constraint. For example, one vertex can be set to lay on a given surface.

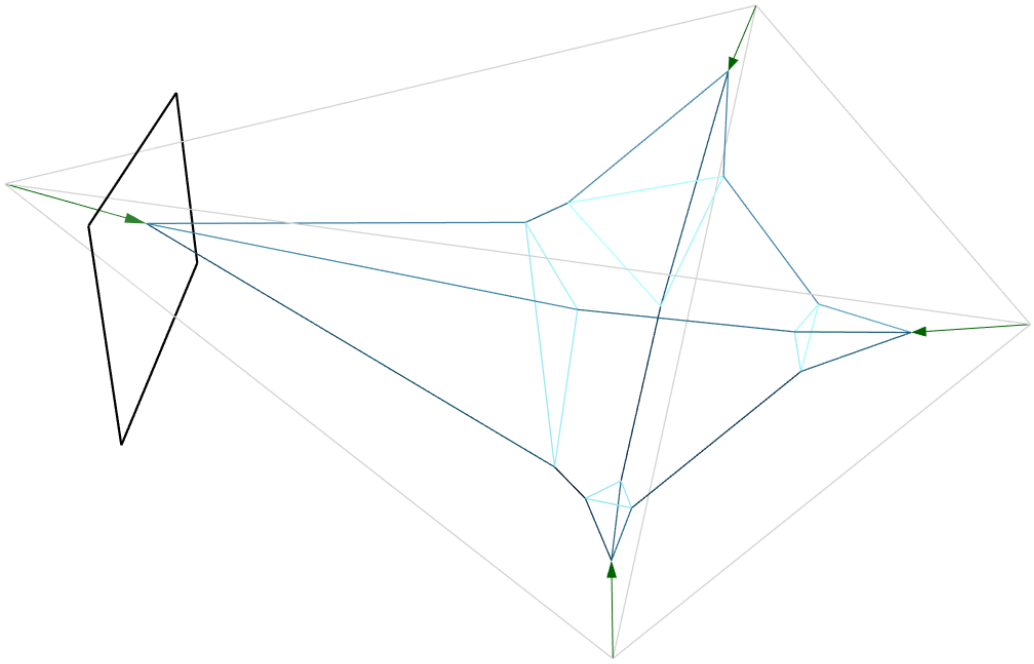
Step.3

Set another vertex constraint. Set one vertex to fixed.



Step.4

Perfom perping with othetwise standard settings. The constrained points should act accordingly

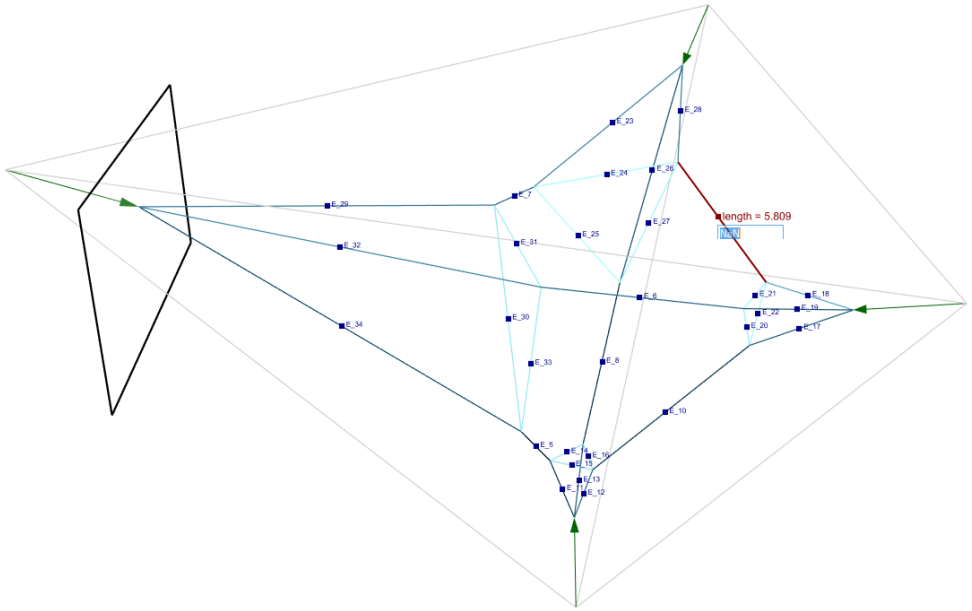


Step.5

Reload the perped model into PFPerp maintaining constraints. All constrained points will keep their limitations.

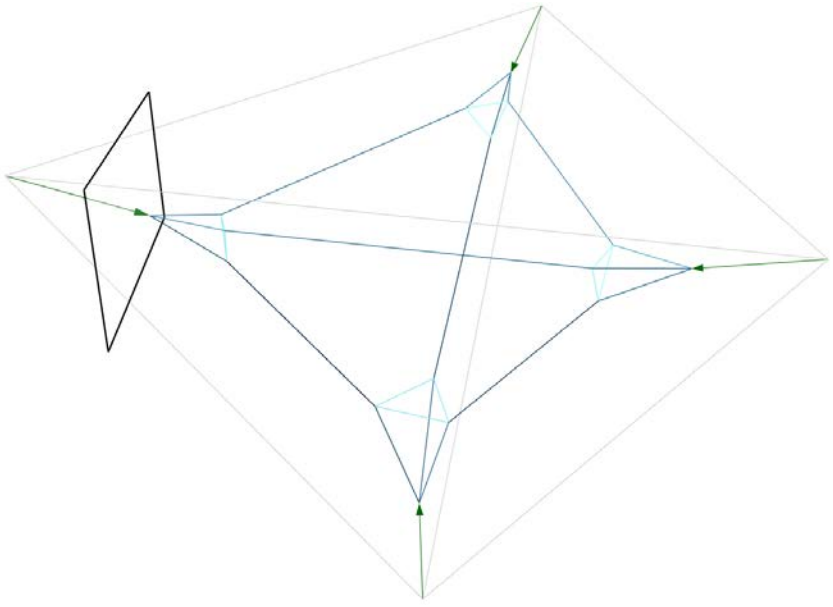
Step.6

Set at least one edge length constraint. For example, set one edge to twice its size



Step.7

Perform another perping round. If the model is not fully perped this step can be repeated with a higher number of iterations.



References

M. Akbarzadeh, T. V. Mele, and P. Block, “On the equilibrium of funicular polyhedral frames and convex polyhedral force diagrams”, *Computer-Aided Design*, vol. 63, pp. 118–128, 2015.

M. Akbarzadeh, “Three Dimensional Graphical Statics using Polyhedral Reciprocal Diagrams,” *dissertation*, Zurich, 2016.

A. Nejur, M. Akbarzadeh, “Constrained manipulation of polyhedral systems”, *IASS 2018 : Creativity in structural design*, Boston, 2018.