The fresh stuff!

- On our way to 1.0! (Toponaming, assembly)
- Sketcher UX improvements
- Theming and UI
- FPA, Ondsel and around
Toponaming

The problem:
Toponaming

- The solution comes from the Link branch by @Realthunder
- An engine that remaps and tracks component names
- Almost done!
Assembly

In a moment!
Sketcher UX

Auto-constraining

Automatically selects vertical/horizontal length constraints
Sketcher UX

On-screen input

Allows to insert dimensions on creation
Theming and UI
Theming and UI
FPA, Ondsel and around

The community is growing

- We have our own non-profit org! The FPA
- Commercial player developing for FreeCAD: Ondsel
- Getting inspiration from Blender
Ondsel Assembly Solver

Aik-Siong Koh
2024-02-04 Sun
freeCAD by askoh

- Basic 3D CAD with Motion Simulation
  - https://www.ar-cad.com/
- Visualworks Smalltalk and OpenGL
- Used as addin in Alibre, SpaceClaim
OndselServer

- Assembly constraints for FreeCAD.org
- Smalltalk motion simulator translated to C++
- https://ondsel.com/blog/
- https://github.com/Ondsel/Development/MbDTheory
- https://github.com/Ondsel/Development/OndselSolver
Assembly Theory
Constraints

• Absolute
  \[ G_{abs} = q_{lp} = 0 \]

• Euler Parameter
  \[ G_E = E_1^2 + E_2^2 + E_3^2 + E_4^2 - 1 = 0 \]

• At Point
  \[ G_{leleO}(q,s,t) = r_{leleO}(q,s,t) = 0 \]

• In Plane
  \[ G_{lele}(q,s,t) = r_{lele}(q,s,t) = 0 \]

• Perpendicular
  \[ G_{perp}(q,s,t) = n_{leO}^T t_{JeO} = 0 \]

• Distance
  \[ G_{lele} = r_{lele} - f_{lele}(t) = 0 \]

• Constant Velocity
  \[ G_{\dot{q}}(q,s,t) = i_{leO}^T j_{JeO} + j_{leO}^T i_{JeO} = 0 \]

• Coupler
  \[ G_{nlc}(q,s,t) = G_{nlc}(r_{ilele}, \theta_{ilele}, ...) = 0 \]
## Joints

<table>
<thead>
<tr>
<th></th>
<th>Rigid (no motion)</th>
<th>Prismatic (1)</th>
<th>Revolute (1)</th>
<th>Parallel Cylinders (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cylindrical</strong></td>
<td>(2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Spherical</strong></td>
<td>(3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Planar</strong></td>
<td></td>
<td>(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Edge Slider</strong></td>
<td></td>
<td></td>
<td>(4)</td>
<td></td>
</tr>
<tr>
<td><strong>Cylindrical Slider</strong></td>
<td>(4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Point Slider</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Spherical Slider</strong></td>
<td></td>
<td></td>
<td>(5)</td>
<td></td>
</tr>
<tr>
<td><strong>Crossed Cylinders</strong></td>
<td></td>
<td></td>
<td></td>
<td>(5)</td>
</tr>
</tbody>
</table>
Smalltalk to C++ Translation

• Simplified C++
  • Very Smalltalk like
• Public and Virtual methods
• Use Smart Pointer std::shared_ptr
  • Pointer with reference counting
  • No memory leak worries
  • No new or delete
  • No difference in passing by value or reference
  • Need to avoid circularity
Digital Twin Concept (2002)

The Digital Twin

Engineering
- Drawings
- Specification
- BIM Model

Operations
- IoT Feeds
- Sensors
- Smart Appliances
- Maintenance
- Occupation
- Energy

Information
- Asset Locations
- Asset Details
- Product Details
- Maintenance Regimes
- Inspections

Data Storage
- Performance indicators
- Analytics
- Dependency
- Human interface
Digital Twin Concept

Digital Twin $\equiv$ Modeling Simulation Animation
Digital-TwinS: Digital Twin applied to Software

- Combine best of static and dynamic languages
- TIOBE Index (Dec 2022) popularity ranking
  1. Python (dynamic)
  2. C (static)
  3. C++ (static)
Why Digital-TwinS

Input → C++ is FAST, complex → Output

C++ is best of FAST

Input → Py is FLEXIBLE, simple → Output

Python is very FLEXIBLE
Digital-TwinS: C++ and Python

Input $\Rightarrow$ C++ Production Program $\Rightarrow$ Output

Input $\Rightarrow$ Python Twin Program $\Rightarrow$ Output

Same Input Same Output (SISO)
Internals can be independent
Same Input Same Output (SISO)

Twins can be any size or any component
Internals can be partially dependent
C++ is FAST at all cost

Python is NIMBLE and rugged
Low cost

A hybrid vehicle would have compromised capabilities
Java, C#, Obj C
C++ Heavy Infrastructure
Small area

Python Light Infrastructure
Large area

Execution

Exploration

We want to win in both settings
Why Digital-TwinS cont.

Input → C++ is Machine centric → Output

Input → Py is Programmer centric → Output

Humans think Objects

“Development at the speed of thought”
Why Digital-TwinS cont. 1

Input → C++ for Computer Experts → Output

Input → Py for Domain Experts → Output

Synergy and feedback between experts
Python and FreeCAD for Brain Dump
Why Digital-TwinS cont. 2

• Assume developing a brand-new feature.
• Python alone can do it in T days. But the feature is slow.
• C++ alone can do it in 5T days. But the feature is fast.
• Twins can do it in 3T days. Python development T days. Guided port to C++ is 2T days. Feature is fast and development is shorter.
• Twins cross-checking each other will reduce bugs in both greatly. This is a bonus.
Strategy for Digital-TwinS

• Capture C++ algorithms in Python twin
  • Executable documentation

• Experiment in Python twin (superset program)
  • Fearless programming

• Transfer discoveries to C++ twin
  • Manually, automated or both
  • Strict testing
  • Iterate with twin

• Debug in Python twin

• Transfer fixes to C++ twin
Prototype vs Digital-TwinS vs Production

Prototype Dev is Nimble and Simple in Py

Production Dev is Difficult in C++. Runs FAST

Prototype in Py  Digital-TwinS in Py  advanced features

Production in C++  stable features

Production in C++  Slow progress  Faster progress  stable features

Digital-TwinS in Py  advanced features