Welcome

Keys to Success with KBE

Dave Cooper, Genworks International

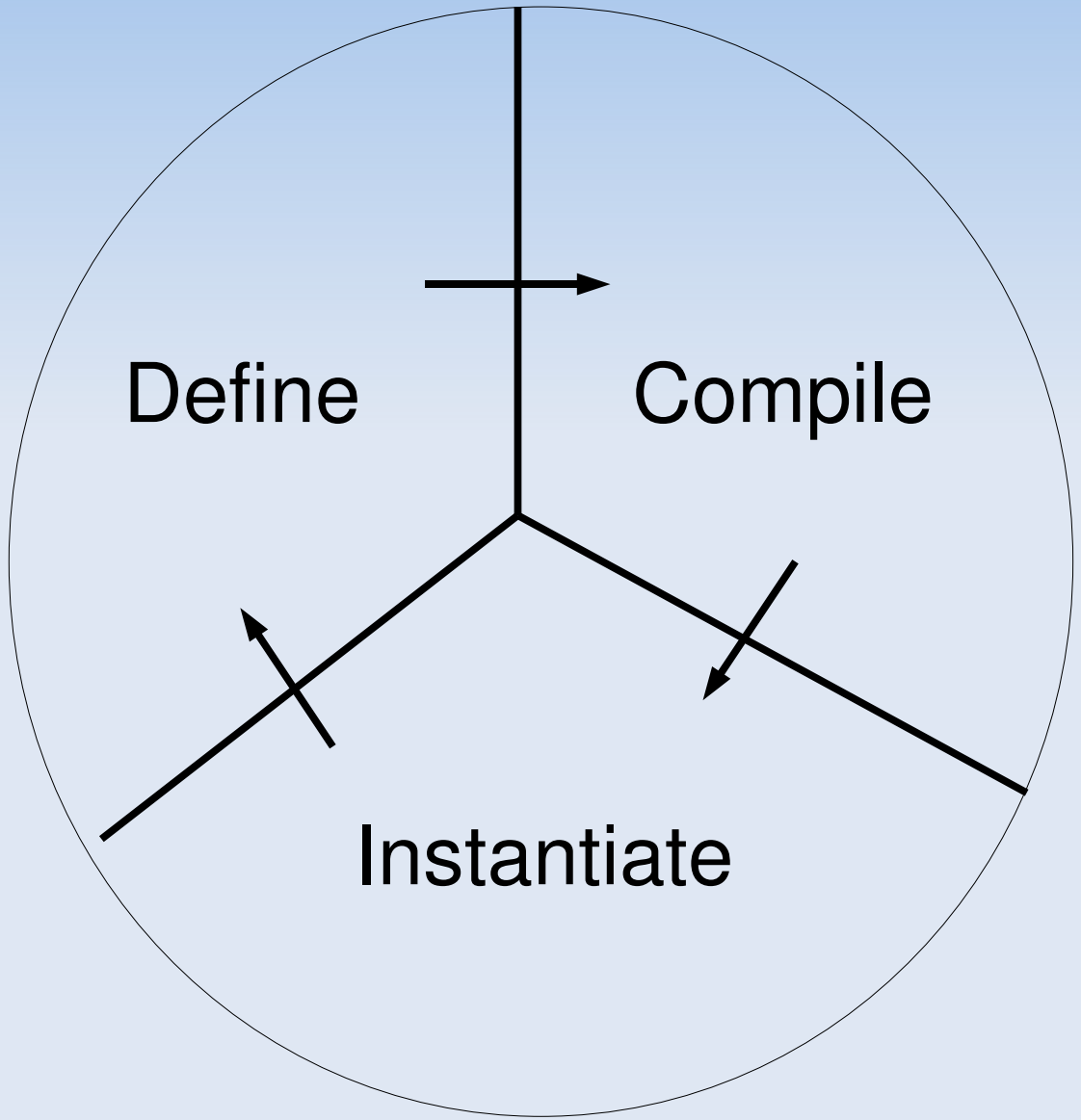
SAE Aviation Conference, August 2008

Outline

- The Cooper Definition of KBE
- Fundamental KBE Features
- Overall Goals of KBE in 2008 and 21st century
- Features ”beyond the basics” to meet these Goals
- Overview of current Tools Marketplace

KBE (The Cooper Definition)

1. The User can **define** any Object using Declarative prose
2. The System can **compile** this definition into efficient machine code
3. The System can **instantiate** Objects and compute Results from the compiled definition



Standard KBE Features

1. Dynamic Language
2. Sophisticated Compiler involving **many levels** of code transformation (macroexpansion)
3. Runtime value *caching* and *dependency-tracking*

Goals for KBE in 21st Century

- Ease and Speed of Application Development
- Ease and Speed of Deployment to End Users
- Longevity of Application Code

Features for Today's Needs

- Declarative, Dynamic Language Environment
  - Well-established, continuing to evolve
  - Expect continued evolution, no magic bullet here
- Web-centric Development and Deployment
  - Embedded webserver
  - Web Browser as User Interface
- Standards-based Modeling Language
  - Ensures compliance of basic building blocks
  - Higher-level framework follows de-facto standard

Web-based Development IDE

GENWORKS INTERNATIONAL

Genworks Ta2: UpWind 5MW Ref - Mozilla Firefox

File Edit View History Bookmarks Tools Help

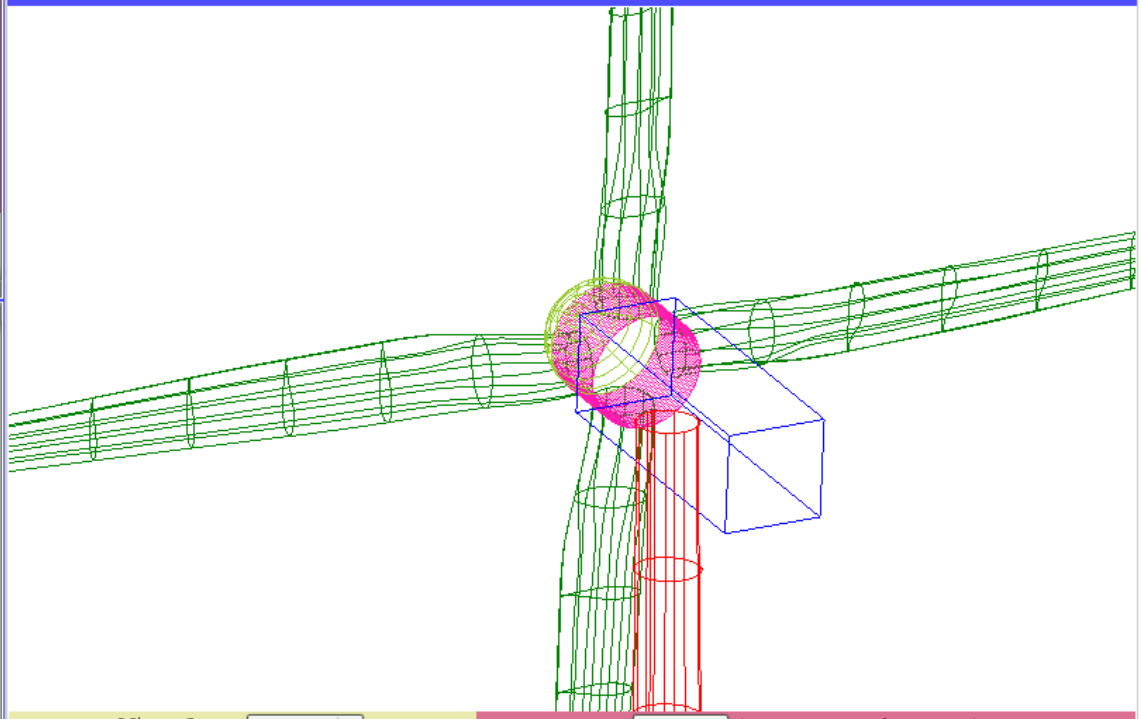
http://localhost:9000/sessions/192cc095e93/index.html

- UpWind 5MW Ref
  - Rotor-Assembly
    - Blade 1
    - Blade 2
    - Blade 3
    - Blade 4
    - Turbine-Hub
      - Hub-Cylinder
      - Hub-Arc
      - Hub-Con 0
    - Turbine-Nacelle
      - Ucs-Nacelle
        - Axe 0
        - Axe 1
        - Axe 2

root?	*Unbound*
rotor-assembly	#<wind::rotor-assembly @ #x764d
rotor-hidded	#<wind::blade @ #x764de552>
safe-children	(#<wind::rotor-assembly @ #x764de522> #<wi
saved-slots	*Unbound*
show-surfaces?	*Unbound*
show-title?	*Unbound*
standard-saved-slots	*Unbound*
standard-views	*Unbound*
strings-for-display	"UpWind 5MW Ref"
support-axis-data	(0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0
support-data	("Cylinder" "Cylinder" "Cylinder" "Cylinde
support-hidded	#<wind::support @ #x77126012>
support-radius-data	(0.0 8.76 17.52 26.28 35.04 43.8 52.56 61.
support-thickness-data	(100 100 100 100 100 100 100 100 100 100 1
tilt-angle-data	5

I... DL... DN... AL... AL*... DL... AN... SR... UN... B... UI... Done.

CL! U! UR! RR! *Default* Default

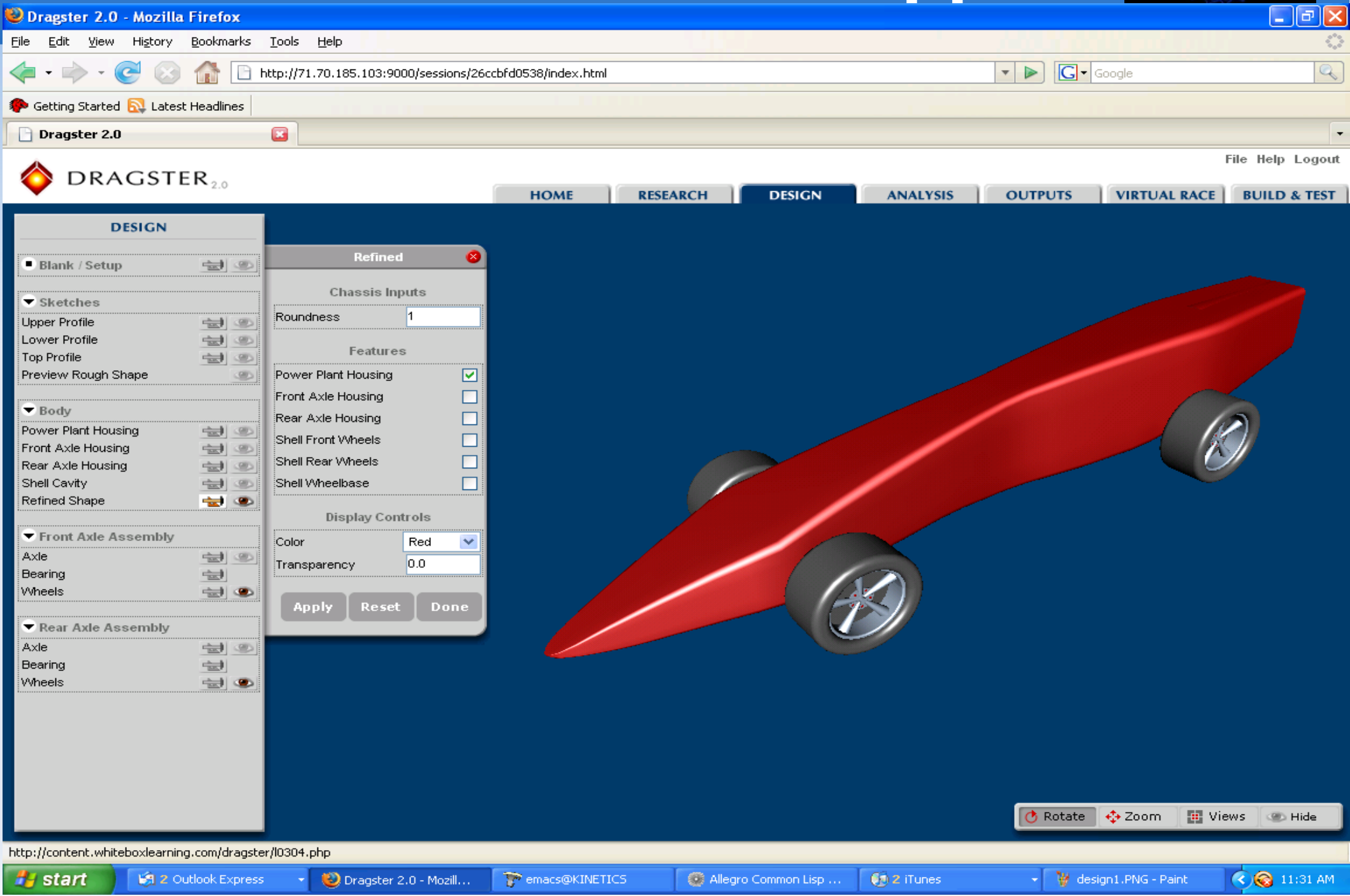


View from Trimetric Restore (unzoom and unpan)

Zoom Factor: 2 Format Png

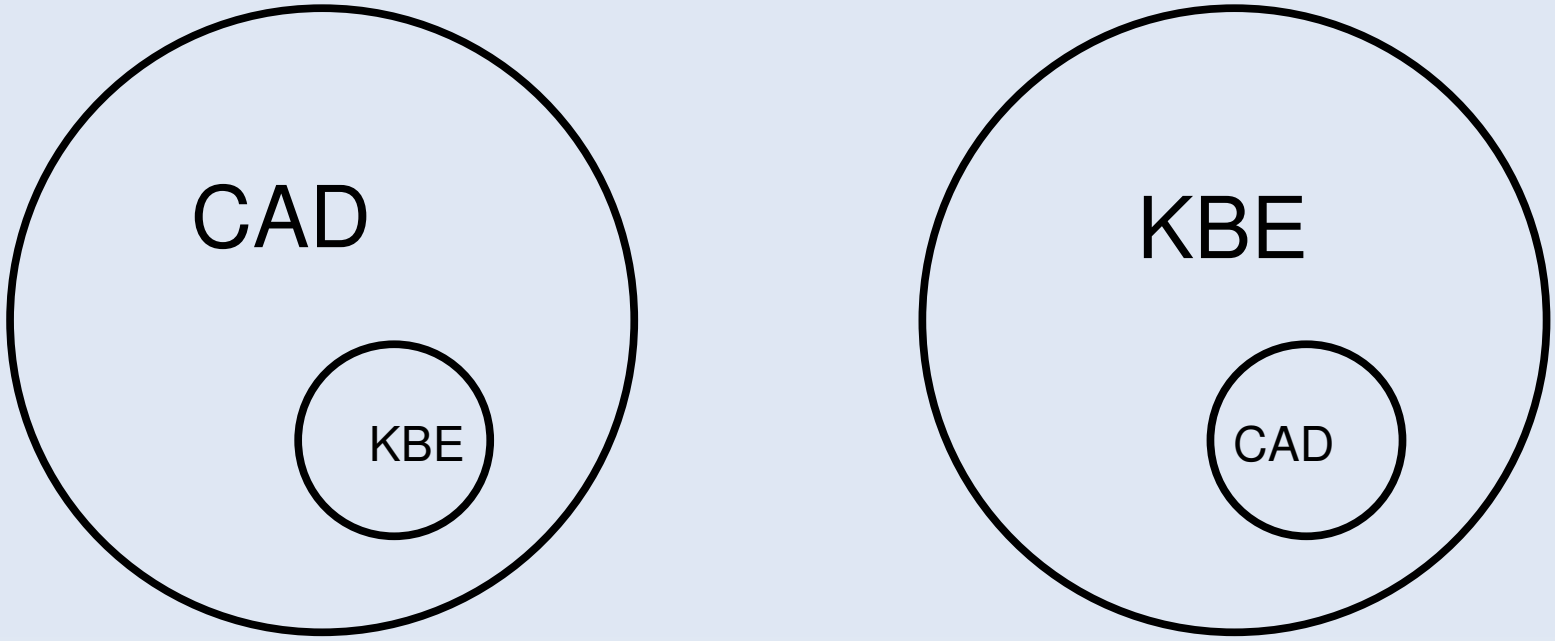
Done

Web-based Runtime App



KBE Tools Marketplace Today

- Two Major Approaches:
  1. CAD Systems with KBE tacked-on, ”KBE-lite.”
  2. Language-kernel based KBE systems (”pure-play” KBE)



CAD augmented with KBE

Examples:

- CATIA Knowledgeware
- UG Knowledge Fusion

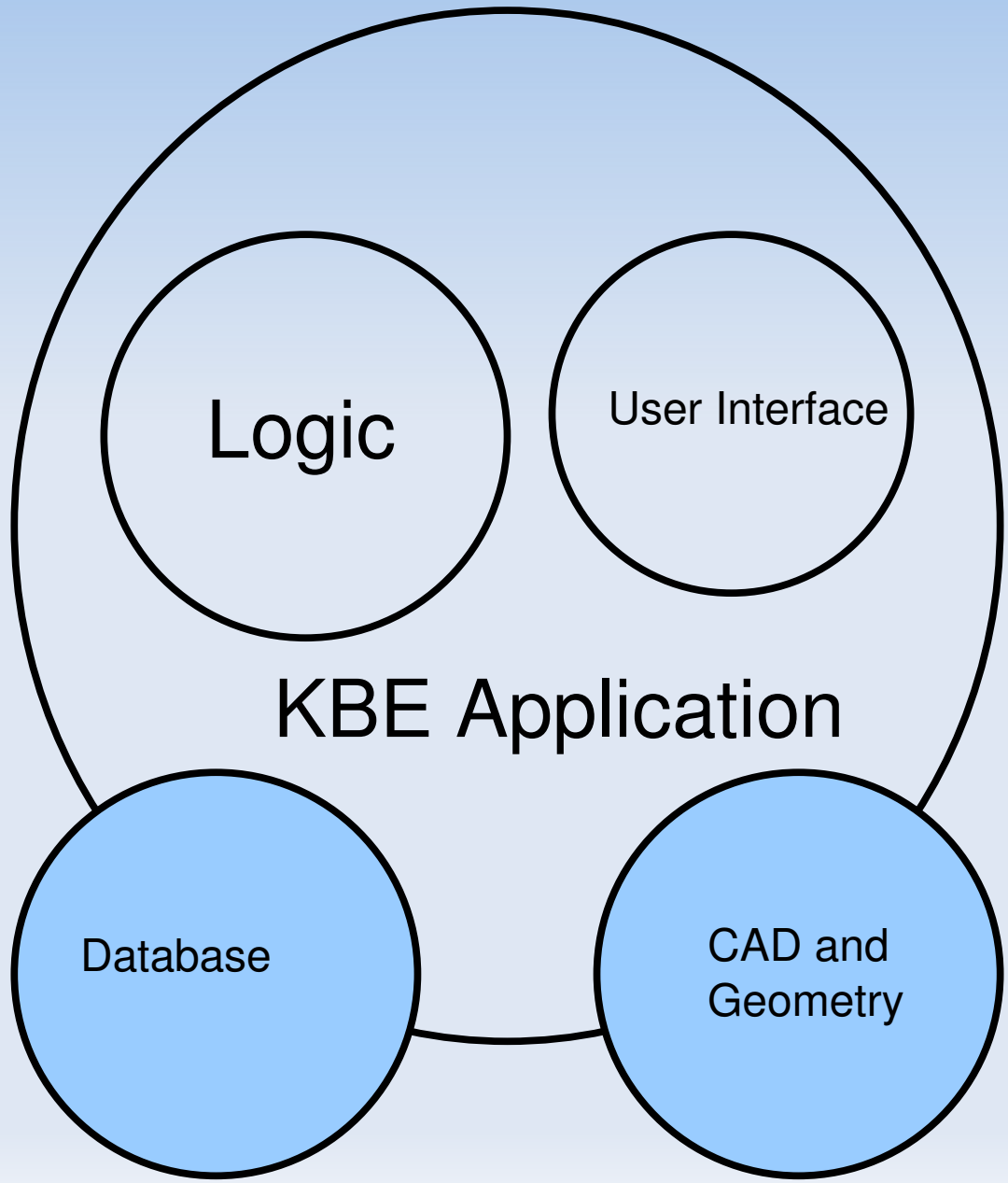
Issues with this approach:

- Entire Desktop CAD system required
- Desktop-based (not web-based)
- High Platform Dependency
- Not fully generative (constrained by CAD model)
- Application Development requires Real Programming

KBE and Geometry

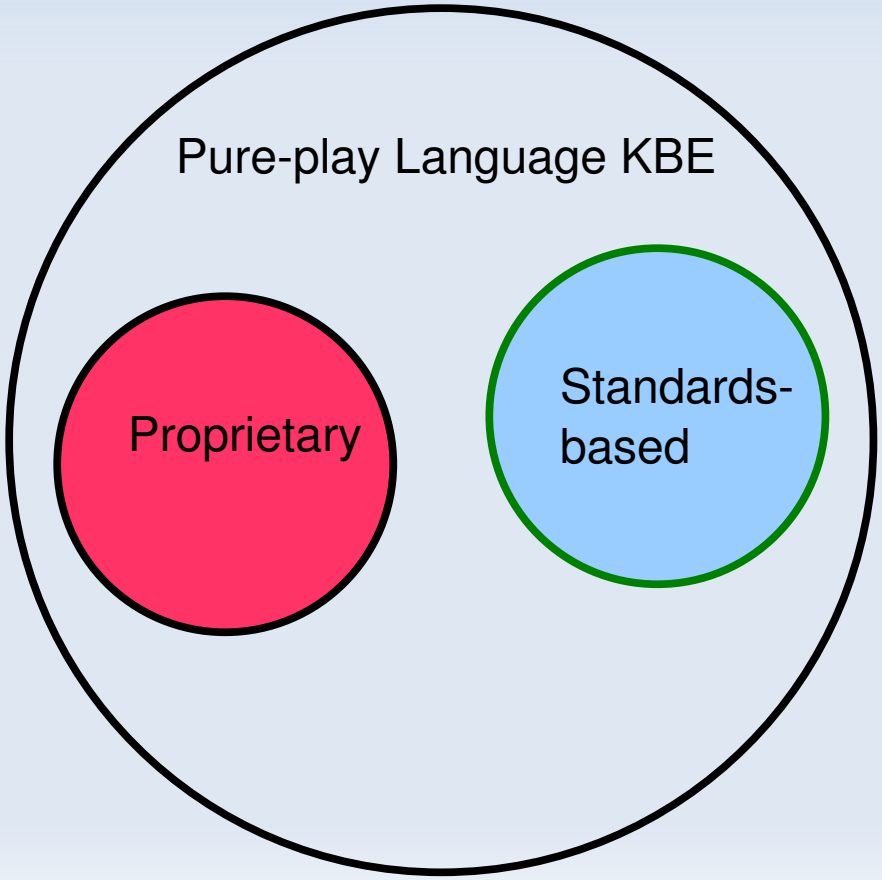
Side Note:

- KBE is more general than CAD
- KBE system must handle geometry, integrate with CAD
- Do not confuse KBE with CAD



Pure-play Language KBE

- Proprietary Language KBE
- Standards-based Language KBE



Proprietary Language KBE

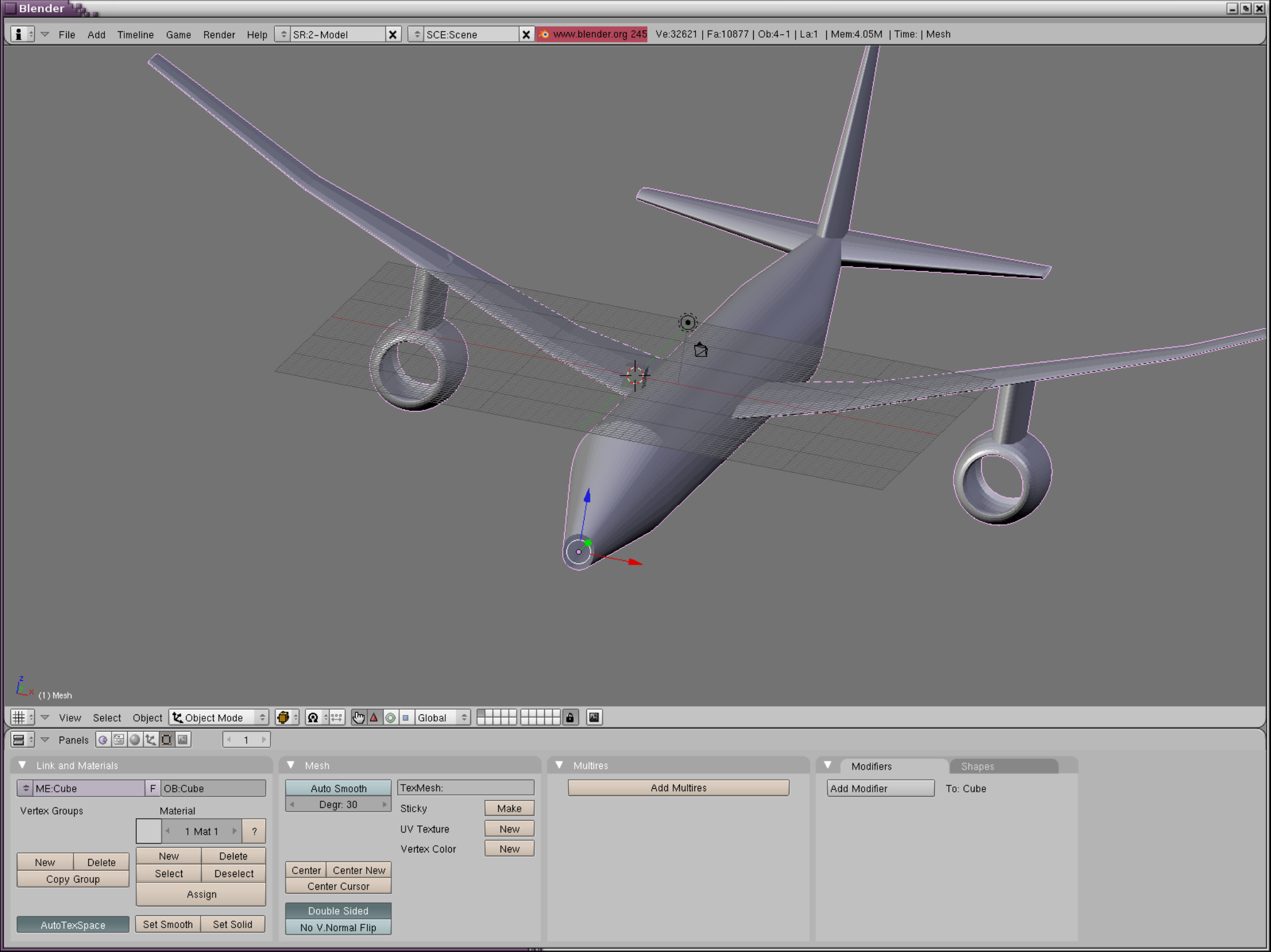
- Language is defined and controlled by a single vendor -- not a superset of any Industry Standard
- User is locked into using that vendor for libraries and extensions
- Language specification can change at, any level, according to vendor's whim

Standards-based KBE

- Core language guaranteed to remain stable
- Users have access to any outside libraries and utilities which also comply with the Standard
- Possible choice of vendors and competition for core language implementation

Genworks GDL as Example

- Genworks GDL, based on ANSI CL Standard
  - Stable standard, with 1000-page specification, since 1995
  - Fundamental language principles from 1958
  - Hundreds of free, open-source libraries
  - Continues to adapt while complying with Standard
  - Core language already supports basic necessary KBE features


Blender
File Add Timeline Game Render Help SR:2-Model SCE:Scene www.blender.org 245 Ve:32621 | Fa:10877 | Ob:4-1 | La:1 | Mem:4.05M | Time: | Mesh
(1) Mesh
View Select Object Object Mode Global
Panels 1
Link and Materials
ME:Cube F OB:Cube
Vertex Groups
Material
1 Mat 1 ?
New Delete
Copy Group
New Delete
Select Deselect
Assign
AutoTexSpace
Set Smooth Set Solid
Mesh
Auto Smooth
Degr: 30
TexMesh:
Sticky Make
UV Texture New
Vertex Color New
Center Center New
Center Cursor
Double Sided
No V.Normal Flip
Multires
Add Multires
Modifiers
Shapes
Add Modifier
To: Cube

Genworks Ta2: solids-of-revolution - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:9000/sessions/3aecc0938ab/index.html

freewrl freezes firefox

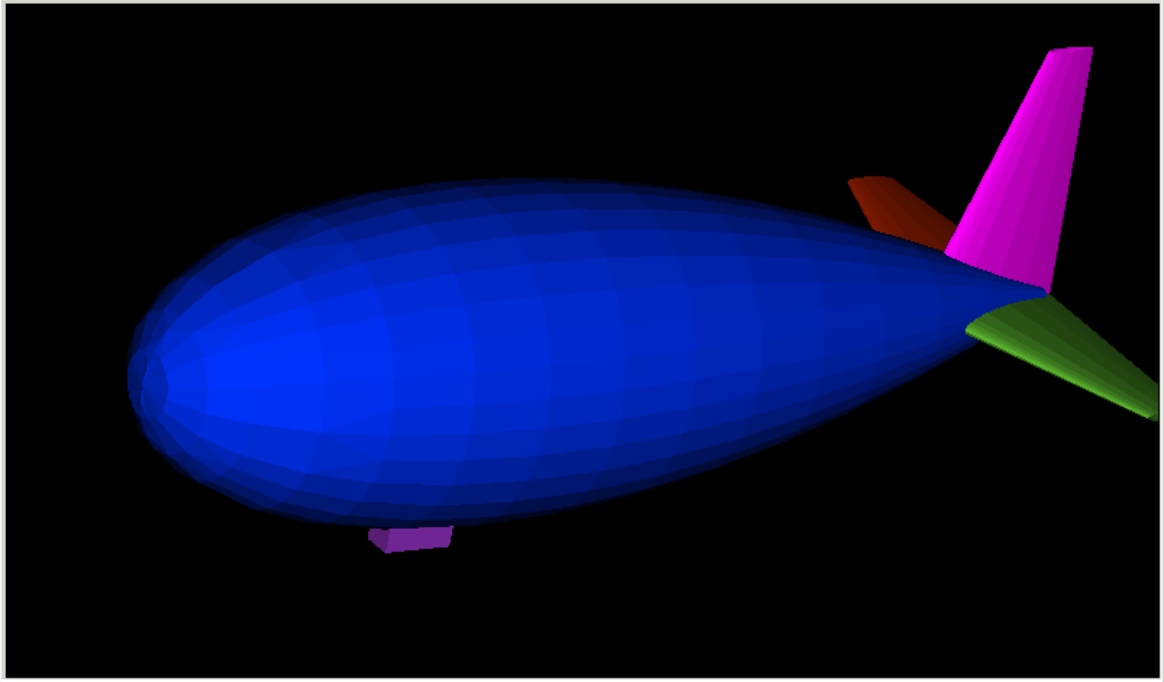
- solids-of-revolution
  - Revolved
  - Stabilizator
    - Horizontal 0
      - Trunk-Hor
        - Loft
    - Horizontal 1
      - Trunk-Hor
        - Loft
    - Vertical
      - Loft
  - Deck

L C R A

I... DL... DN... AL... AL*... DL... AN... SR... UN... B... UI... Done.

CL! U! UR! RR! *Default*

Navigate Preferences Help



Format Vrml View from Top Zoom Factor: 1 Field of View: 1 Apply

decomposed	*Unbound*
density	1
direct-mixins	*Unbound*
display-controls	nil
display-iso-curves-wireframe?	*Unbound*
display-tesselation-lines-wireframe?	*Unbound*
edges	#<gdl::standard-sequence @ #x75e(
end-caps-on-brep?	*Unbound*
faces	*Unbound*
first?	*Unbound*
from-iges?	nil
height	*Unbound*
hidden-children	*Unbound*
iges-level	*Unbound*
index	nil
iso-curves	*Unbound*

Transferring data from localhost...

Aircraft Electrical Connector

TUDelft

Pin Assignment Design Application

Full Update! | Break | TaTu

Selection Mode: source

Selected Pin:
nil

Selected Pin Signal Type:
nil

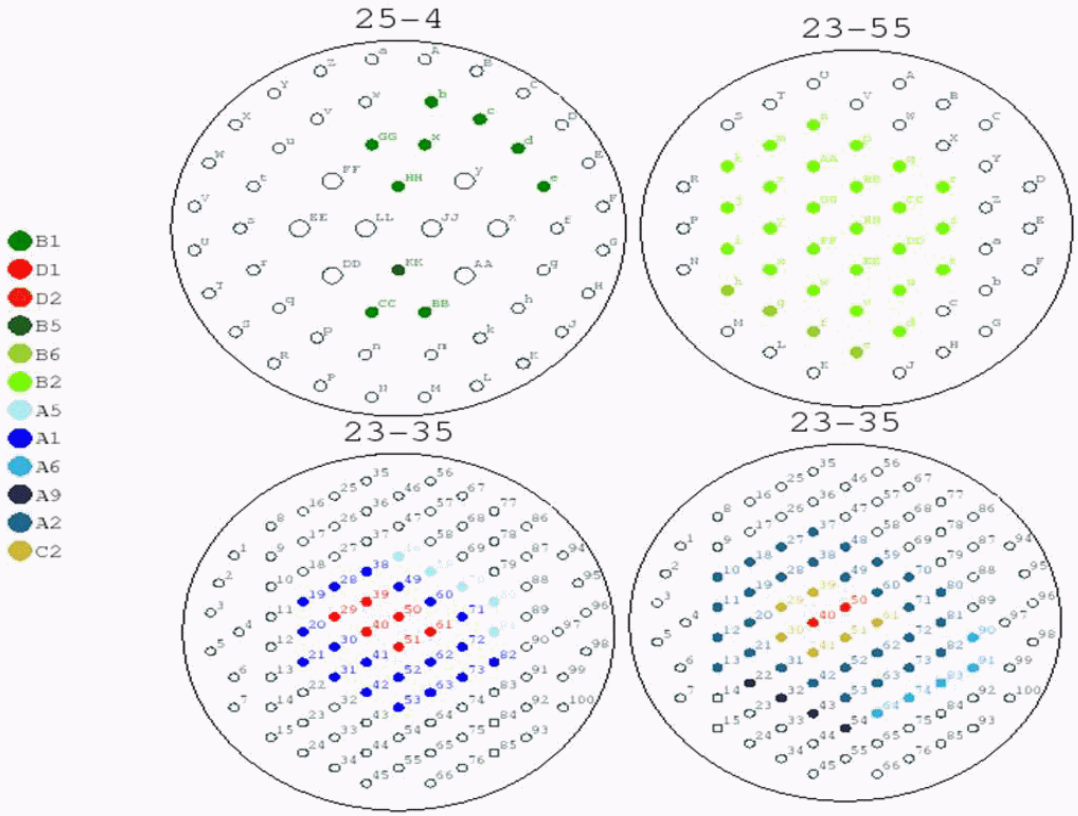
Connector Data: Browse...
Currently Using: \DOCUME~1\SWGVAN~1\LOCALS~1\Temp\connector-data-M13.txt

Signal Data: Browse...
Currently Using: \DOCUME~1\SWGVAN~1\LOCALS~1\Temp\signal-data-M13.txt

Group Radially?

Report Type: Tree Structure

OK

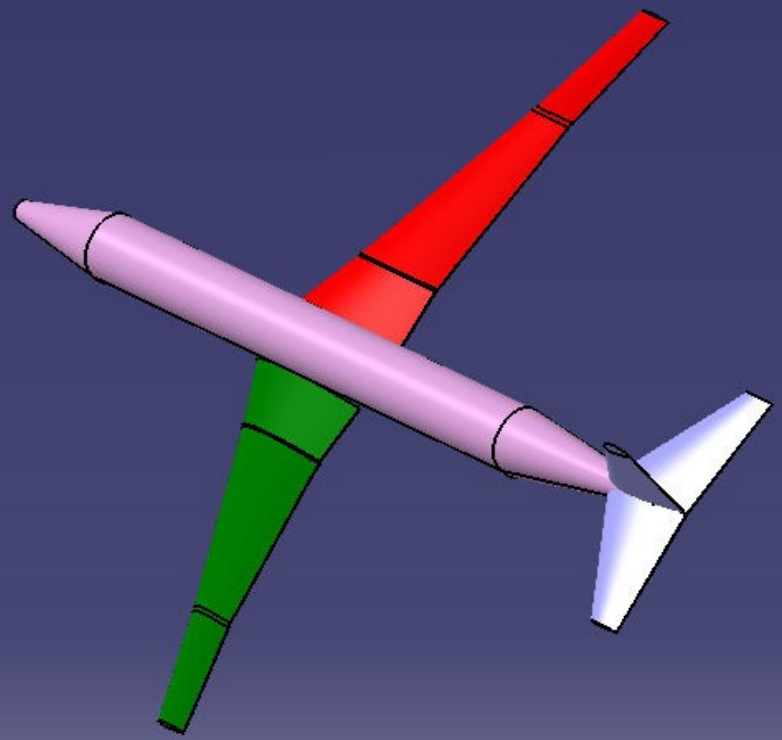


Done

CATIA V5 for Student - [aircraft-assembly.igs]

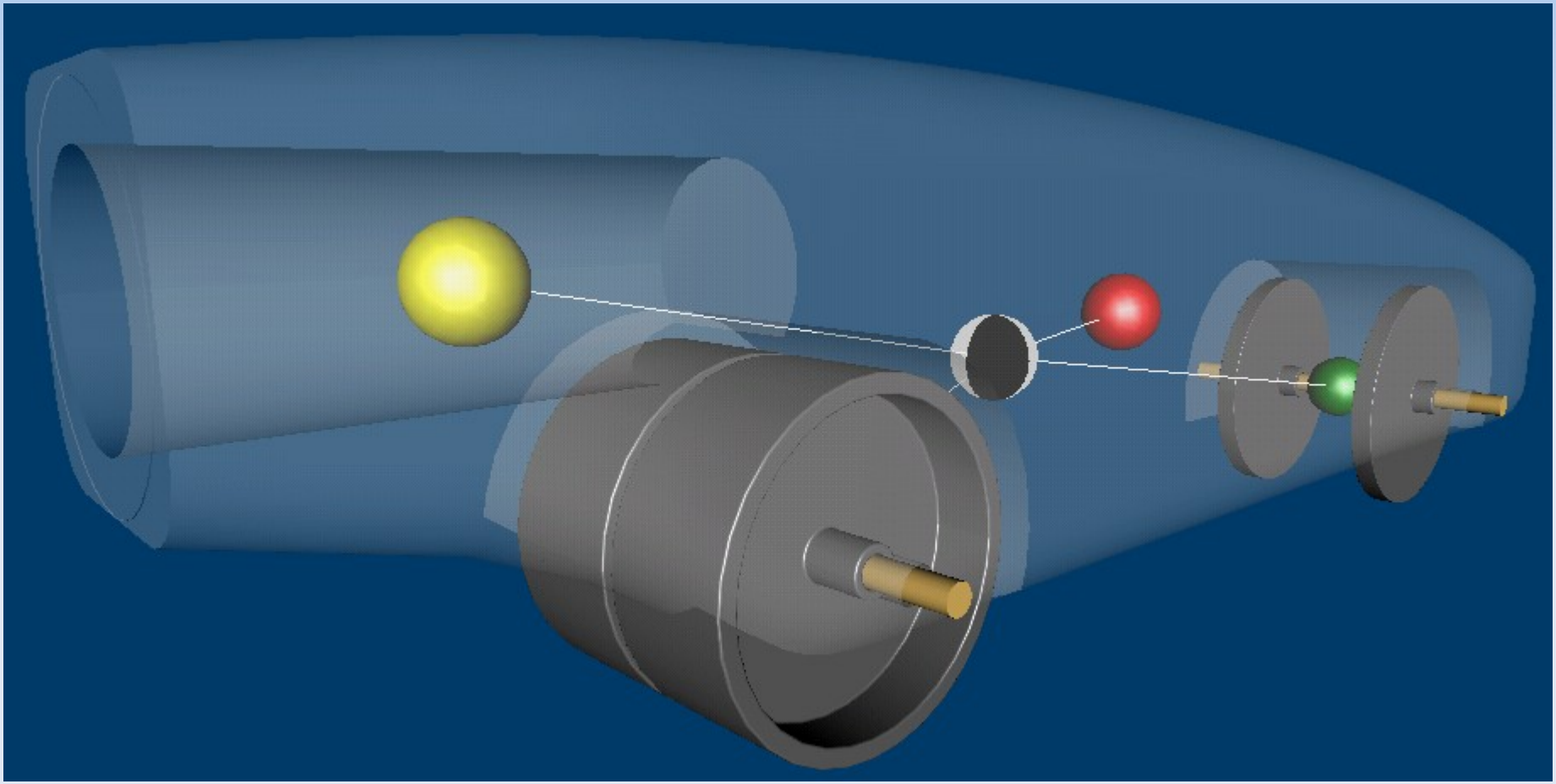
Start File Edit View Insert Tools Window Help

- aircraft-assembly
  - xy plane
  - yz plane
  - zx plane
  - PartBody
  - Geometrical Set.1
    - Surface.1
    - Surface.2
    - Surface.3
    - Surface.4
    - Surface.5
    - Surface.6
    - Surface.7
    - Surface.8
    - Surface.9
    - Surface.10
    - Surface.11
    - Surface.12
    - Surface.13
    - Surface.14
    - Surface.15
    - Surface.16

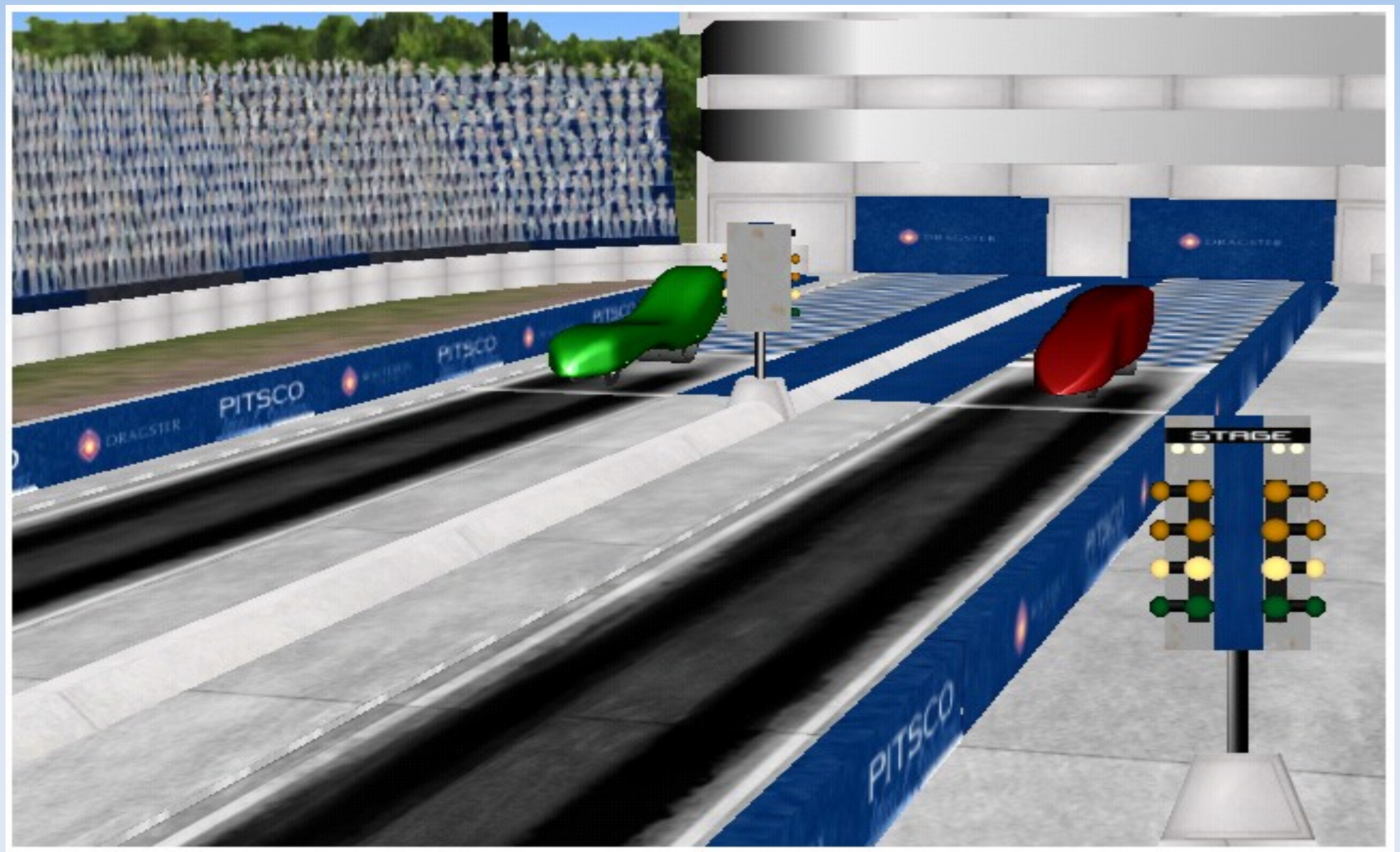


Select an object or a command

Sample Outputs



Sample Outputs



Conclusion - 21st century KBE

The tools are here today to achieve the Goals:

- Exponential Speedup for developing – provided by Dynamic, Declarative language
- Ease of Deployment – provided by web-centric system
- Longevity of Application Coce – provided by Standards-based core language

Discussion

- Author's Contact Information
  - Dave Cooper, Product Development Head
  - david.cooper@genworks.com
  - 248 330 2979

Thank you, Willem, and good morning to everyone else.

As you have heard, my name is Dave Cooper, and I am with a Knowledge-based Engineering company (KBE for short), called Genworks, that markets a high-end software modeling tool which is built on KBE principles.

I have worked more or less in exclusively with KBE technologies for roughly the last 15 years, so I am **beginning** to get a glimmer of what KBE actually is, and what it can **really** to for you.

This 20-minute talk is greatly enlarged in a companion paper entitled ”Keys to Success with KBE” which is part of the SAE conference proceedings

Outline

- The Cooper Definition of KBE
- Fundamental KBE Features
- Overall Goals of KBE in 2008 and 21st century
- Features ”beyond the basics” to meet these Goals
- Overview of current Tools Marketplace

My remarks today will be in the following sequence:

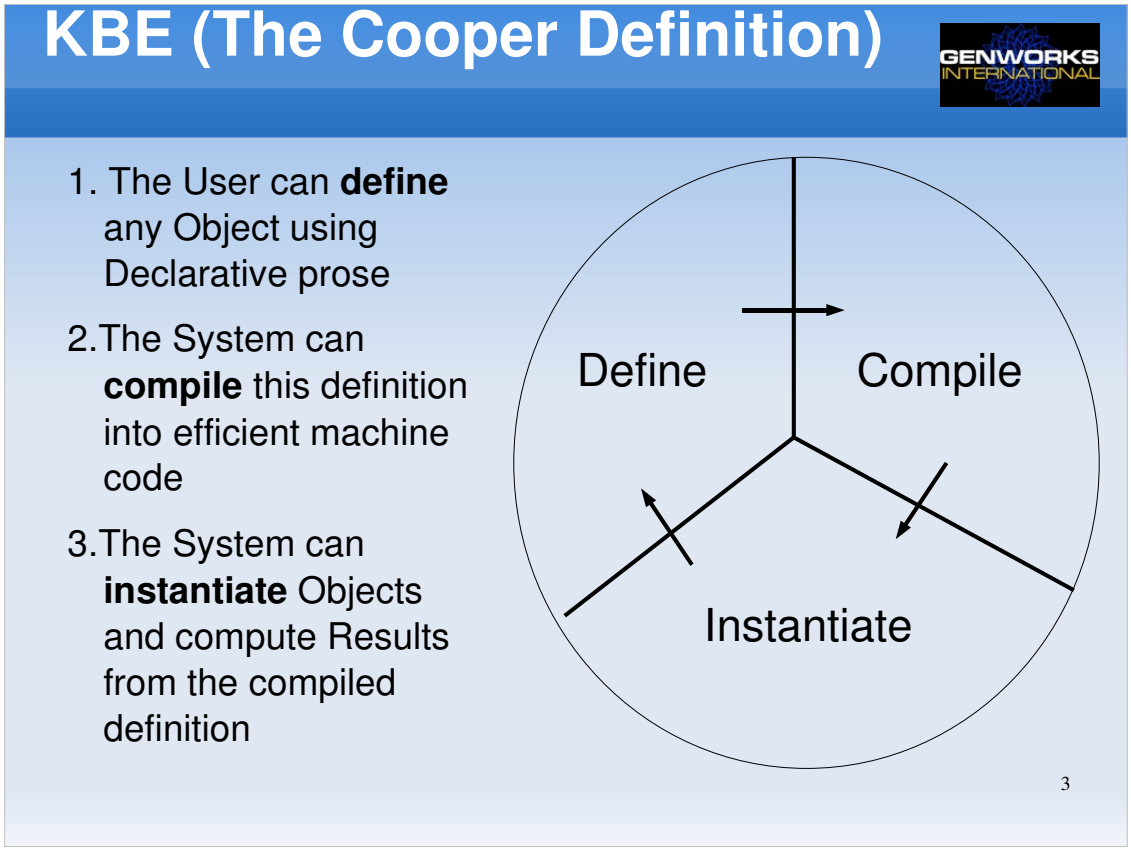
First, what I call the ”Cooper Definition” of KBE

Next, the Necessary standard, or basic features a system must have in order to meet this definition;

Then, a review of the fundamental goals of KBE for today's marketplace;

Along with a look at the necessary additional features a system must have in order to achieve these 21st century goals;

Finally, an overview of the current KBE tools marketplace.



The Cooper Definition of KBE:

A KBE System is a software modeling framework in which:

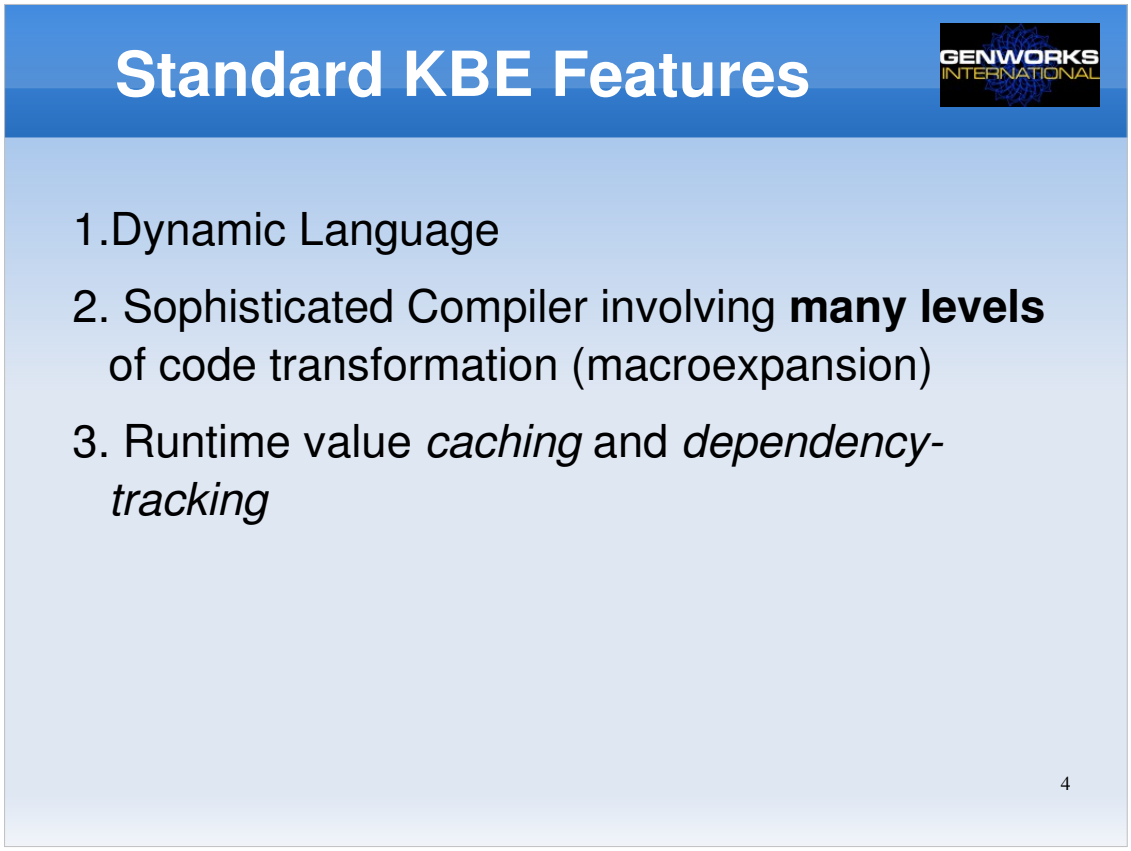
1. A User can define any Object, anything from an airplane wing with complex surfaces to a simple web page, using a Declarative approach.

   Unlike a procedural computer program, a Declarative approach describes *what* something is like, rather than *how* to create it.

   Objects can be defined to contain other objects, and to inherit characteristics from other object definitions.

2. The System can **compile,** or translate, this high-level Declarative definition into lower-level programming language code, and finally into efficient machine code.

3. The Runtime System can create **instances** of the defined objects, run code in the proper order to compute useful results, and keep track of dependencies, automatically. The Runtime system can be a subset of the whole System, with no need for the compiler.



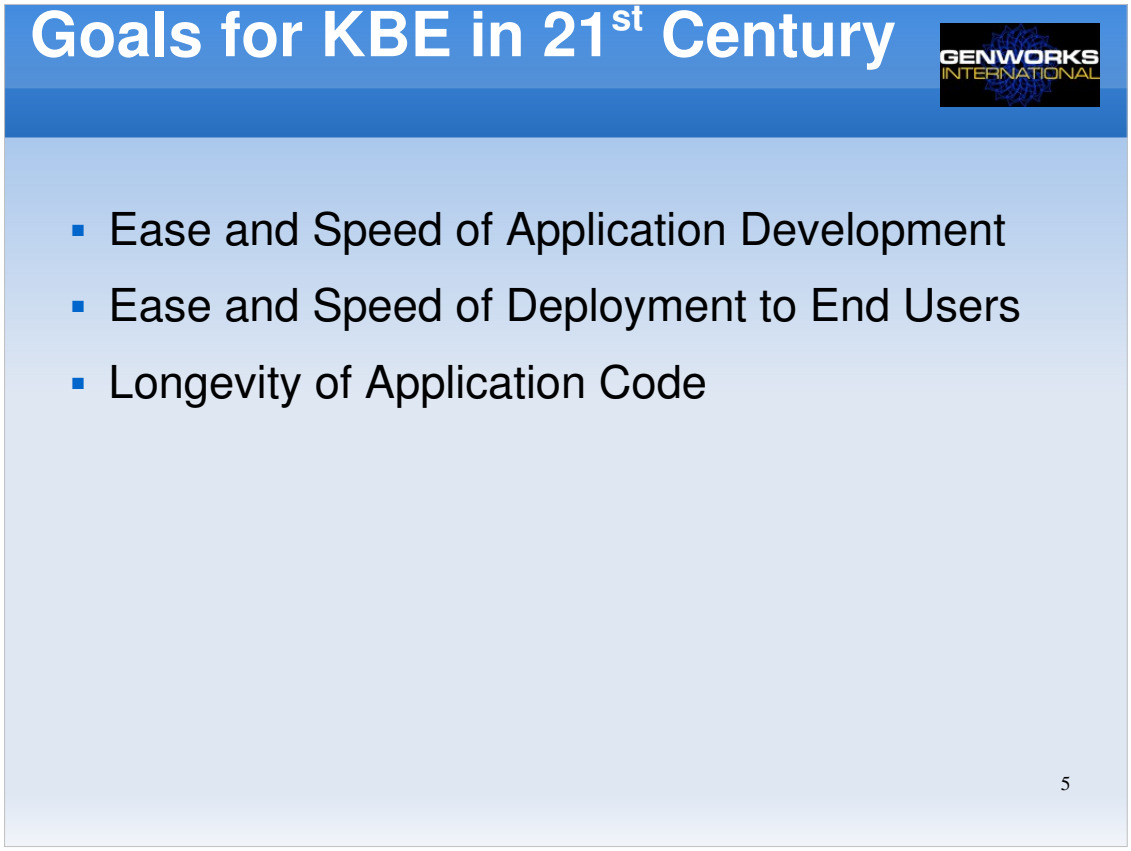
Now I will cover three necessary features (in other words, ”minimum requirements”) which a system must have in order to qualify as ”KBE.” These features roughly correspond to the three items in the Cooper Definition.

First, it needs to be based on a Dynamic Programming Language.

Among many other things, ”Dynamic Language” means that a program (or model) can be altered ”on-the-fly” without having to restart it from the beginning.

Second, the compiler has to have code expansion, or *macroexpansion* capability – this is necessary for pre-processing the extremely high-level Declarative definitions into the low-level code, which can then be compiled into actual executable instructions (assembly language, machine code, etc).

Finally, the system must be able to remember computed results (”caching”), and detect when they are ”stale” and need to be recomputed (”dependency tracking”). This is necessary in order to prevent KBE models (which typically can become very large) from repeating the same computations over and over again.

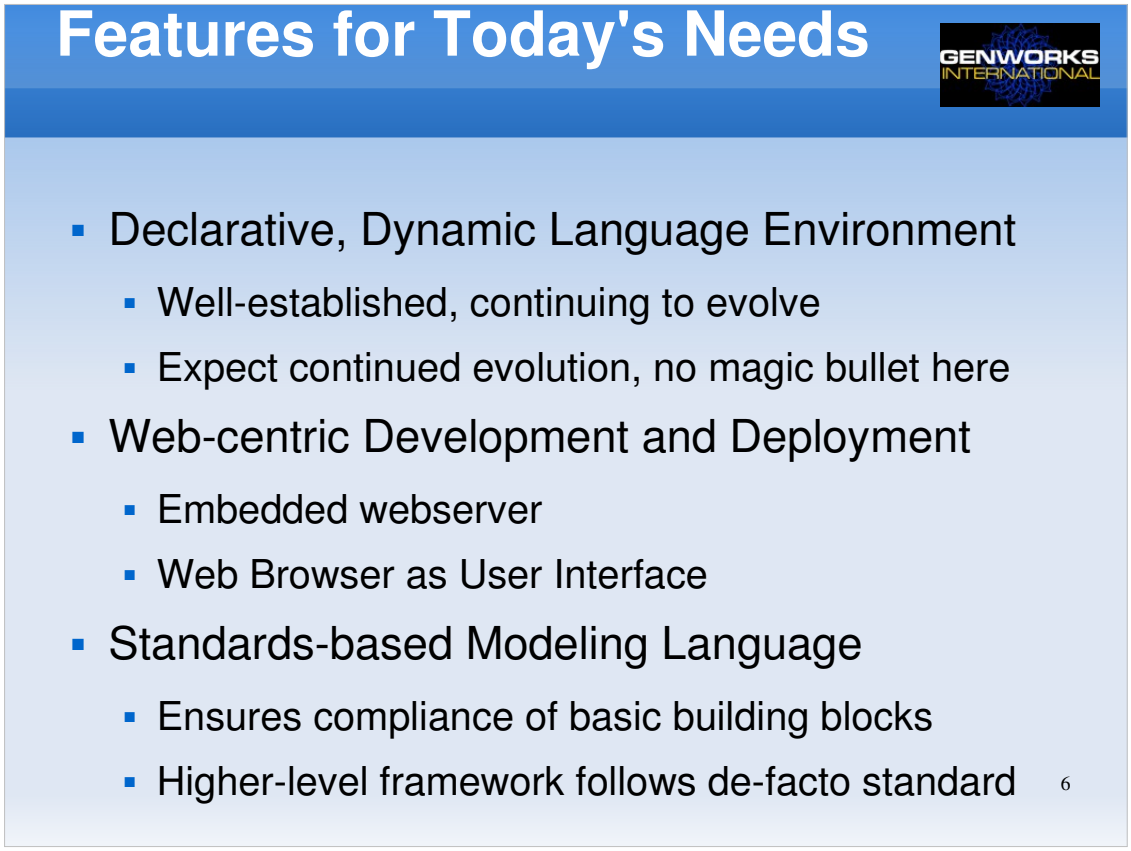


The characteristics and features I have described up to now have been available in KBE systems, ICAD for example, for more than 20 years now.

Now let's take a look at what the goals of KBE should be today, in 2008. Keeping with our pattern of ”threes,” here are three top goals which I feel should always remain in primary focus for any KBE effort:

1. Given a reasonable learning curve, the tool should make it as easy and fast as possible for a KBE developer (”knowledge engineer”) to develop, test, extend, and maintain a KBE application.

2. When an application is ready to produce useful results, the tool should make it as easy and fast as possible to deploy that application for use by end-users anywhere in the world, using any type of computer. We do not assume that these end-users even know what KBE is. They just want an application which can give them useful results.
3. Once an application is developed and stabilized in ”maintenance mode,” the application code should survive many years and decades, across changes in computers, operating systems, CAD tools, etc, without needing to be ”thrown away” and rewritten.



When we look at these three Needs of Today, we find that the first, ”Ease and Speed of Application Development,” is already covered by the Standard KBE features.

The whole thrust and purpose of the Declarative language, and everything supporting it is to maximize developer productivity, while minimizing development time and code volume.

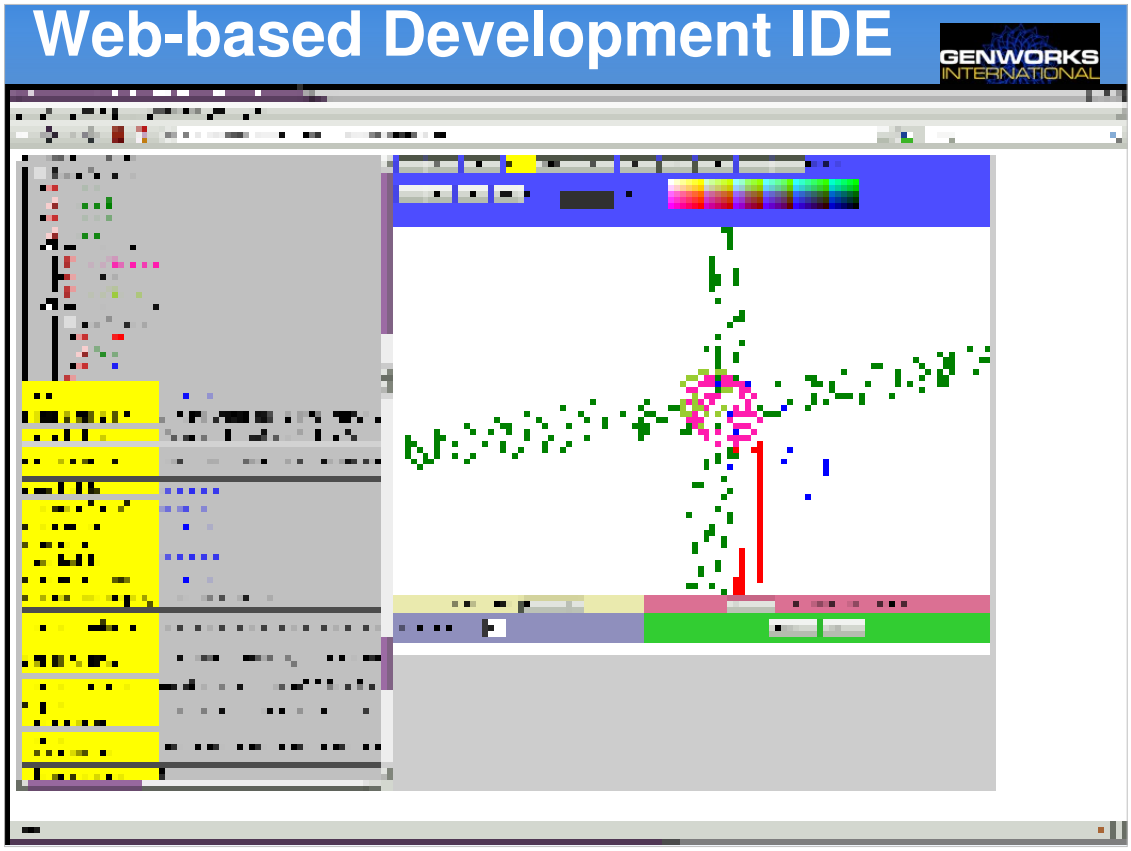
Will these techniques continue to improve? Of course. But fundamentally, the feature is already there and we feel it is worth keeping.

For ease of Application Deployment, however, no other technology comes close to that of the World Wide Web – and Web support is conspicuously absent from our list of ”basic” KBE features.

That is why we feel strongly that, to meet today's needs, a KBE toolkit must embrace the web and provide native support for it.

Finally, the tool needs to support longevity of application code. The key feature here is Standards compliance. The KBE language must fundamentally comply with an established industry standard programming language with a proven track record.

It would be nice to have an Industry Standard for the higher-level declarative KBE syntax as well. But high-level syntax, by its very nature, can be manipulated automatically. The bulk of the complexity lies in the minute details of the low-level primitive operators of the language.



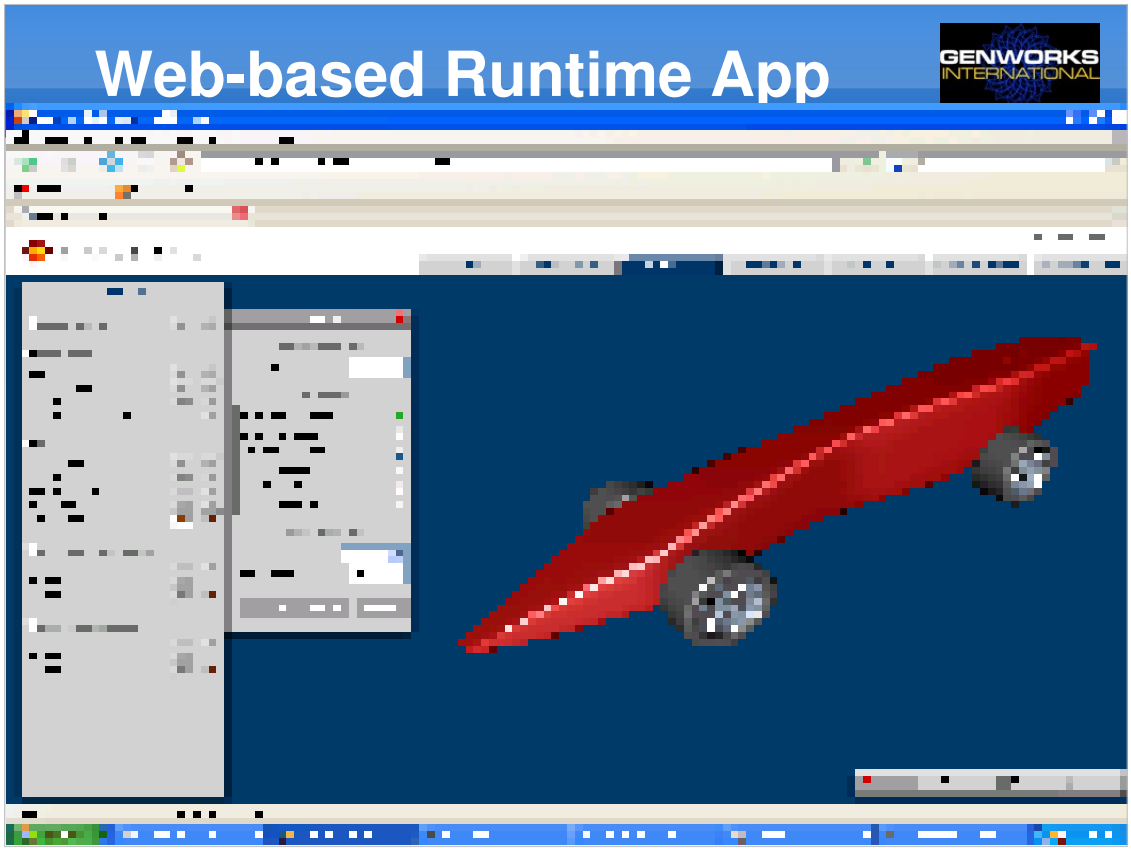
Here is an example of the web-based development environment provided by Genworks GDL.

The developer uses a context-sensitive text editing environment to write and modify the object definitions, then gets instant feedback on the results by using this web-based environment.

Note the expandable object tree it upper-left, Object Inspector in lower-left, and Viewport with Controls in the right part of the screen.

This runs entirely within a standard browser like Firefox or IE, with no need for browser plugins.

It is currently undergoing a major design overhaul at our design studios in Delft.

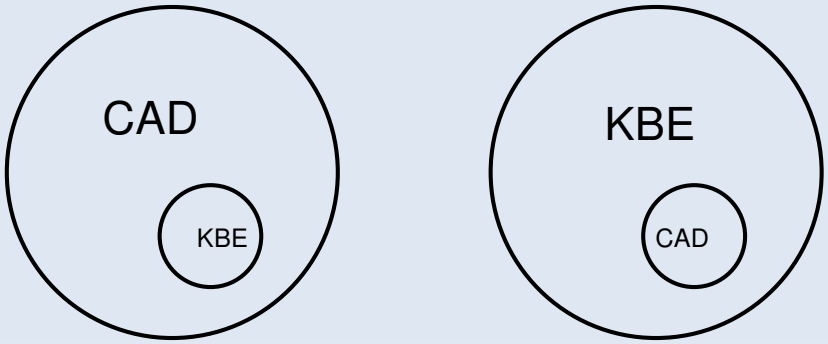


Here is an example of an end-user application, also running entirely within the Firefox web browser.

The end-user sees this as any other web application – he or she does not need to be aware of KBE concepts or the fact that GDL is running in the back end.

KBE Tools Marketplace Today

- Two Major Approaches:
  1. CAD Systems with KBE tacked-on, ”KBE-lite.”
  2. Language-kernel based KBE systems (”pure-play” KBE)



9

In the marketplace today, we see two major categories:

the first are desktop CAD systems at their core, which have been augmented with some capabilities resembling KBE.

The second are KBE systems at their core (”pure plays”). To be useful in a Design Engineering role, of course, these systems must contain complete support for geometric entities as well.

CAD augmented with KBE

Examples:

- CATIA Knowledgeware
- UG Knowledge Fusion

Issues with this approach:

- Entire Desktop CAD system required
- Desktop-based (not web-based)
- High Platform Dependency
- Not fully generative (constrained by CAD model)
- Application Development requires Real Programming

10

The first category, CAD augmented with KBE, looks like it might be a good idea at first, since it promises to avoid translation problems between CAD geometry and KBE geometry.

But on closer inspection, the approach has several long-term issues:

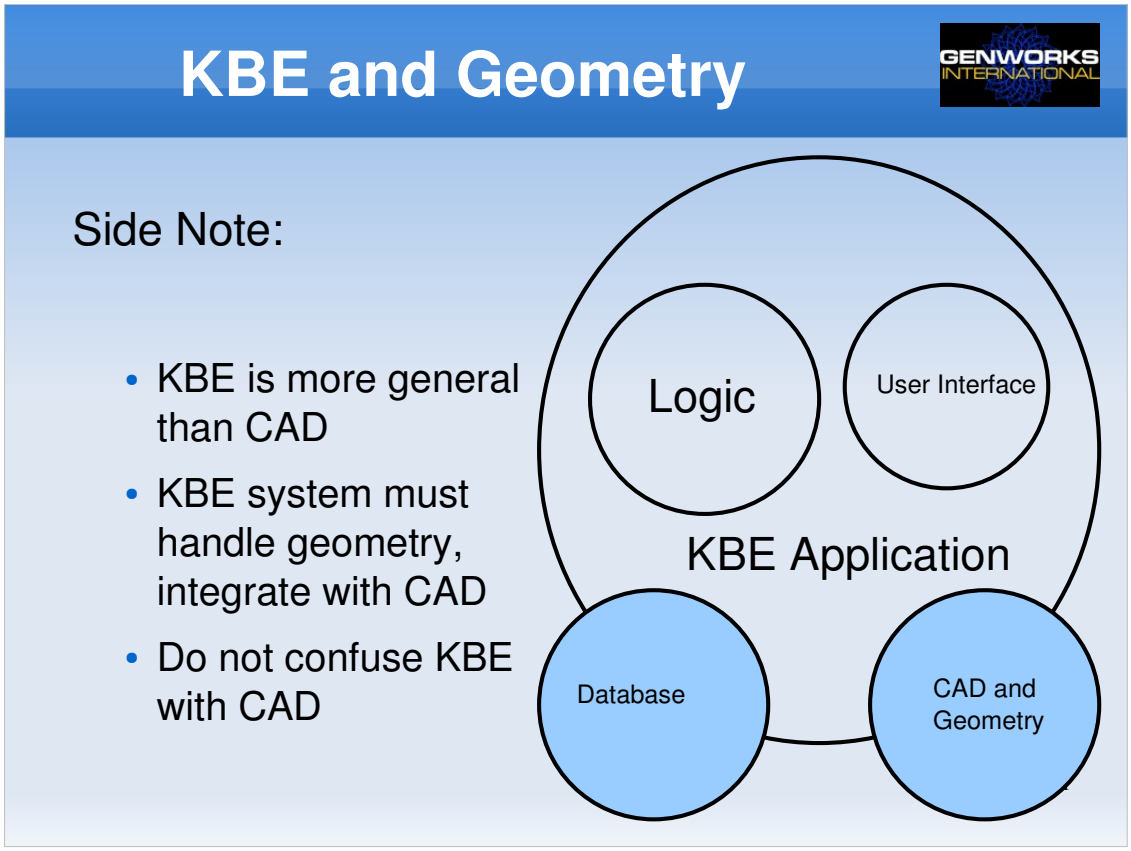
An entire heavy desktop CAD system must be up and running, even to run a small utility application – the approach is basically the opposite of web-centric.

There is not a true language compiler – if there is a KBE language, it is more like a scripting language, run through an interpreter. The lack of a true machine-code compiler can cause severe performance problems for large models.

Models are not generated from generic definition – the ”man behind the curtain” is a traditional CAD model.

Real customization and end-user application development requires ”real” programming in a language like C++.

The approach completely fails the Longevity test. KBE ”rules” developed in this environment tend to be extremely CAD-system specific.
If a company changes CAD systems, or a CAD system changes major versions, detailed KBE code will very likely have to be rewritten from scratch.



I would like to make an important observation on KBE vs CAD.

Because KBE systems and applications can generate and manipulate geometry, they are sometimes confused with CAD systems.

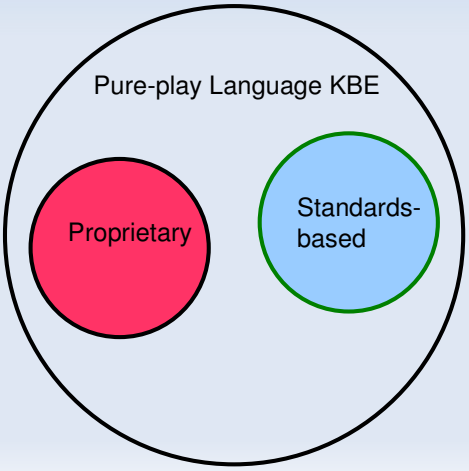
In fact, KBE systems are in a completely different category. They are much more general than CAD systems. KBE systems are used to model, manipulate, and generate all types of information, including numbers, symbols, text, reports, and, yes, geometric entities.

A KBE system needs to integrate with standard geometry, and with CAD systems, in exactly the same way it needs to integrate with databases, web services, and any other part of today's computing infrastructure.

To place KBE in the same ”bucket” with a CAD environment will result in an emphasis on one very narrow aspect of KBE.

Pure-play Language KBE

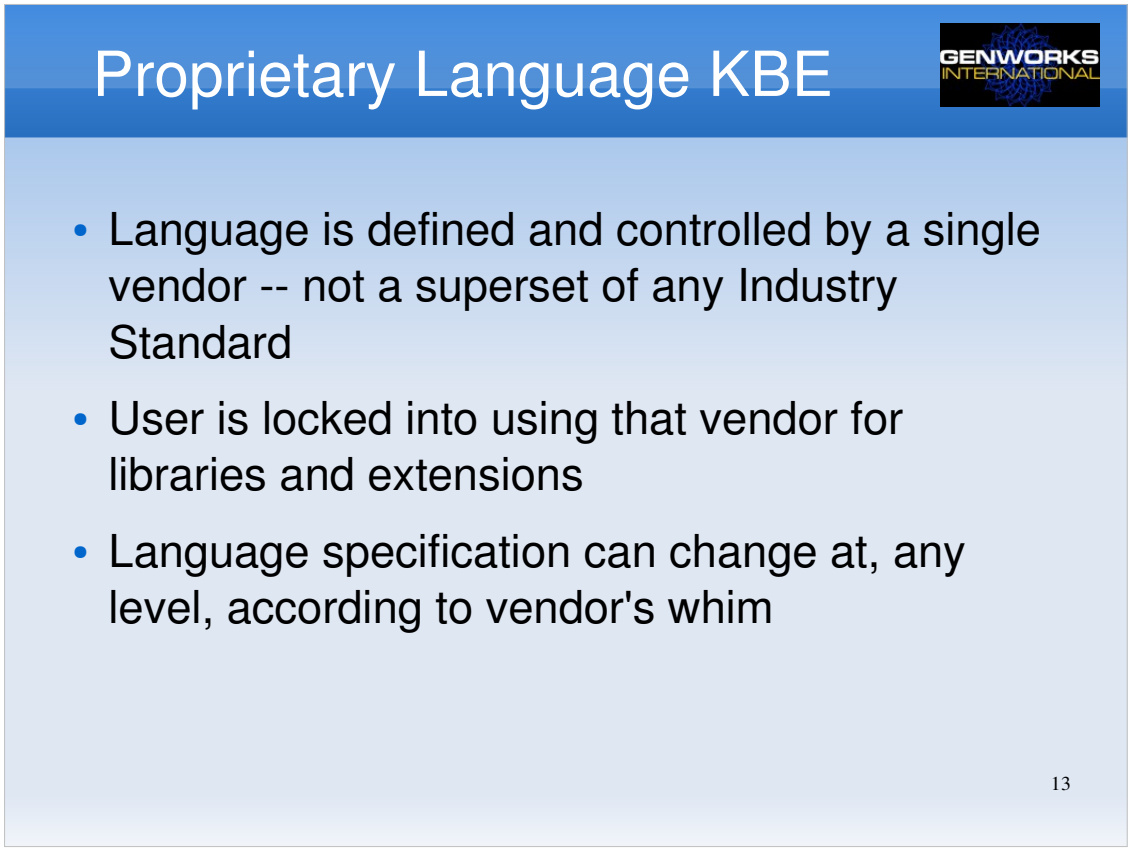
- Proprietary Language KBE
- Standards-based Language KBE



The second main category, the ”pure-play” language-based systems, can also be divided roughly into two categories.

The first is Proprietary Language systems, where a tool vendor has designed and developed a specific Declarative KBE language from the ground up.

The second is Standards-based Language KBE, where a vendor has chosen to use an Industry Standard language as a basis for the Declarative KBE language.



Proprietary Language KBE systems provide a complete development environment in a Declarative language which is not based on or a superset of any Industry standard programming language.

The main issue with this approach is that the user is tied to the ”company store” -- that is, he is locked into depending on that vendor for any libraries and extentions to the language.

Furthermore, the language specification is subject to changes which might break old application code, since it does not have to comply with any outside standards.

Standards-based KBE

- Core language guaranteed to remain stable
- Users have access to any outside libraries and utilities which also comply with the Standard
- Possible choice of vendors and competition for core language implementation

The other approach for a pure-play Language KBE is to base the language on some well-established Industry Standard.

Complying with a Standard might add some burden for the vendor, but this approach has several advantages.

First, the core language is guaranteed to remain stable. This has huge benefits for meeting our goal of Application code Longevity.

Second, when developing an application, the knowledge engineer will be able to use any available outside libraries or software tools which also comply with the Standard.

Finally, because the core language is not defined by a single vendor, there is room for vendor choice and competition in this space.

Genworks GDL as Example

- Genworks GDL, based on ANSI CL Standard
  - Stable standard, with 1000-page specification, since 1995
  - Fundamental language principles from 1958
  - Hundreds of free, open-source libraries
  - Continues to adapt while complying with Standard
  - Core language already supports basic necessary KBE features

15

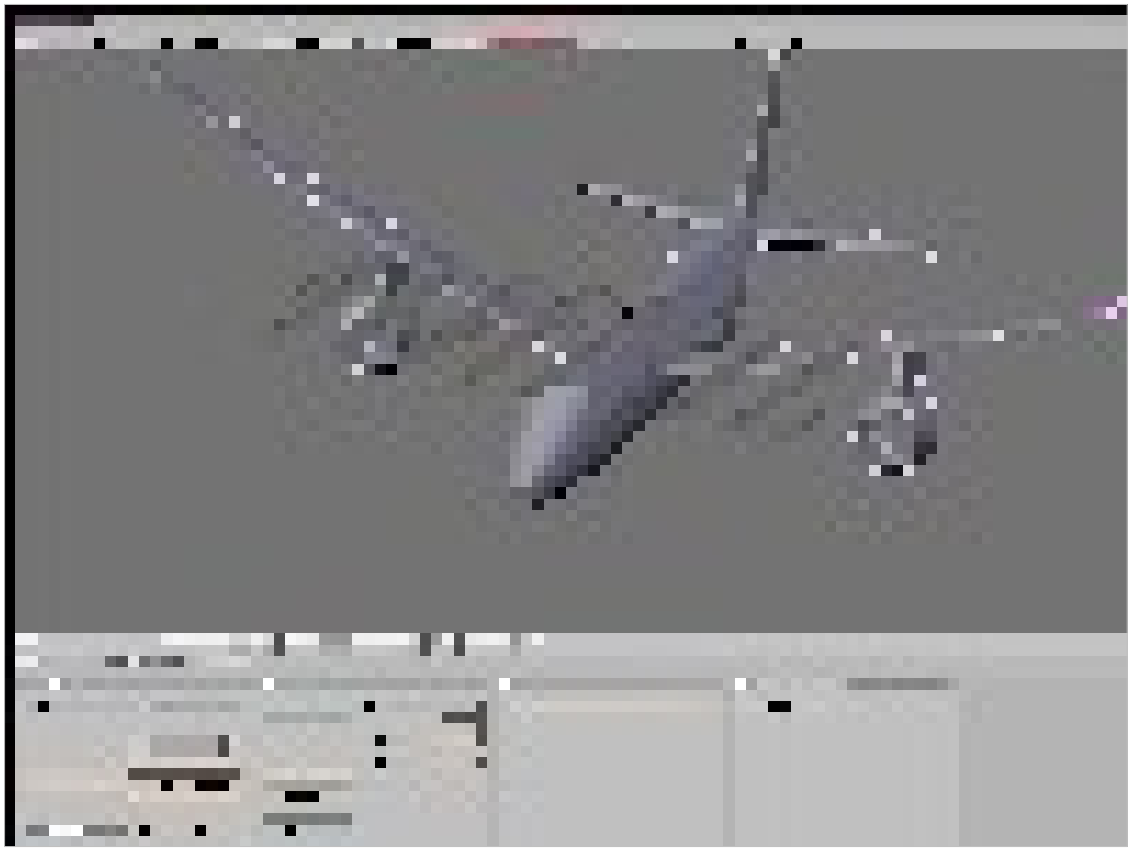
Genworks GDL is a KBE system based on the ANSI CL language standard.

While the standard is stable, hundreds of free, open-source libraries are developed continually by a worldwide community.

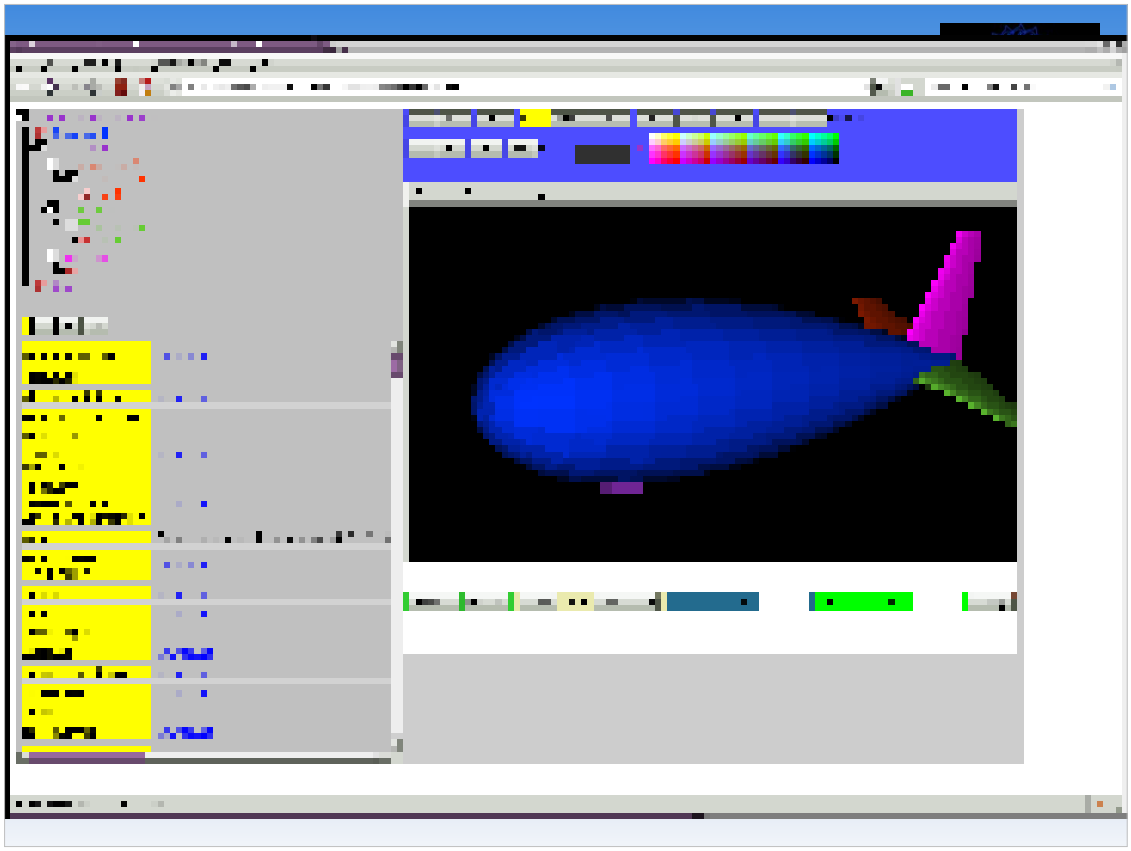
In this way, CL can adapt for ever-changing needs of modern computing, while the core implementations remain compliant with the official Standard.

As it happens, CL also already supplies two of our three Basic KBE required features: it is a highly Dynamic language, and it supports extremely powerful code transformation macros, or macroexpansion.

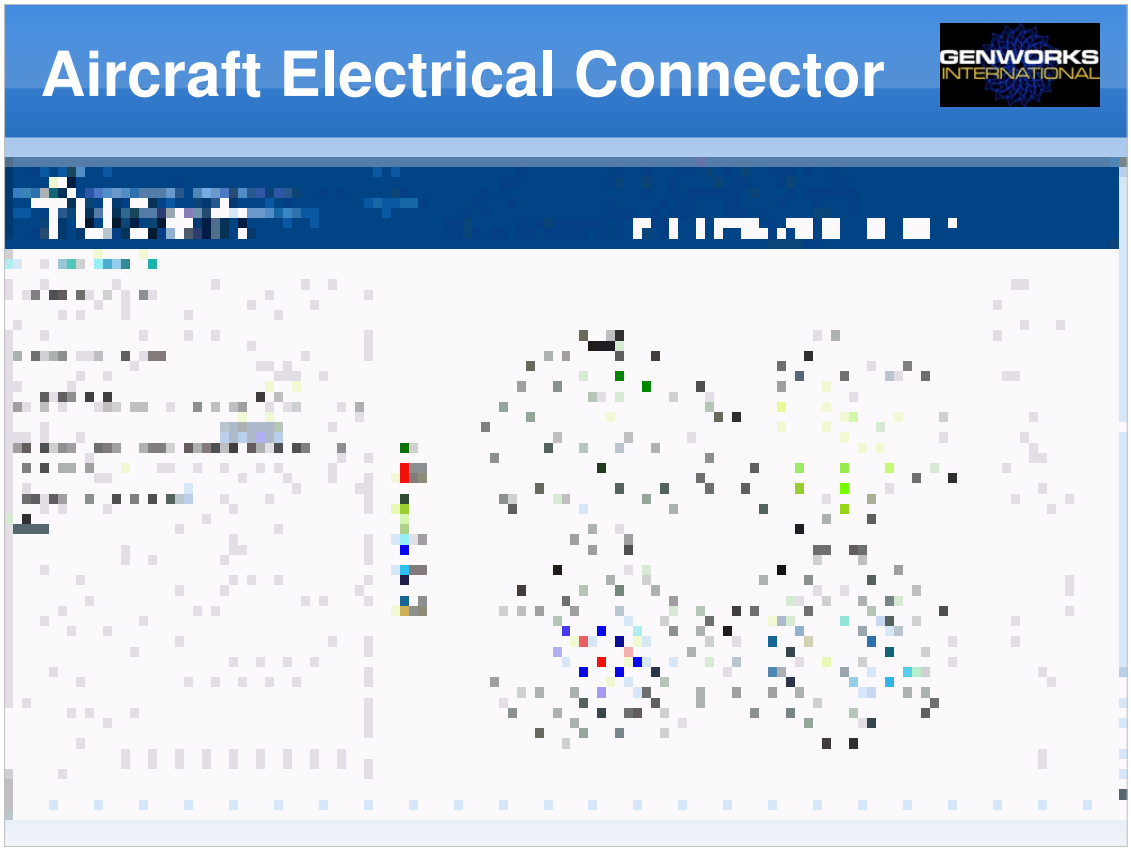
By adopting this standard core rather than re-inventing it, Genworks can focus on making GDL meet the modern KBE requirements we discussed a few moments ago.



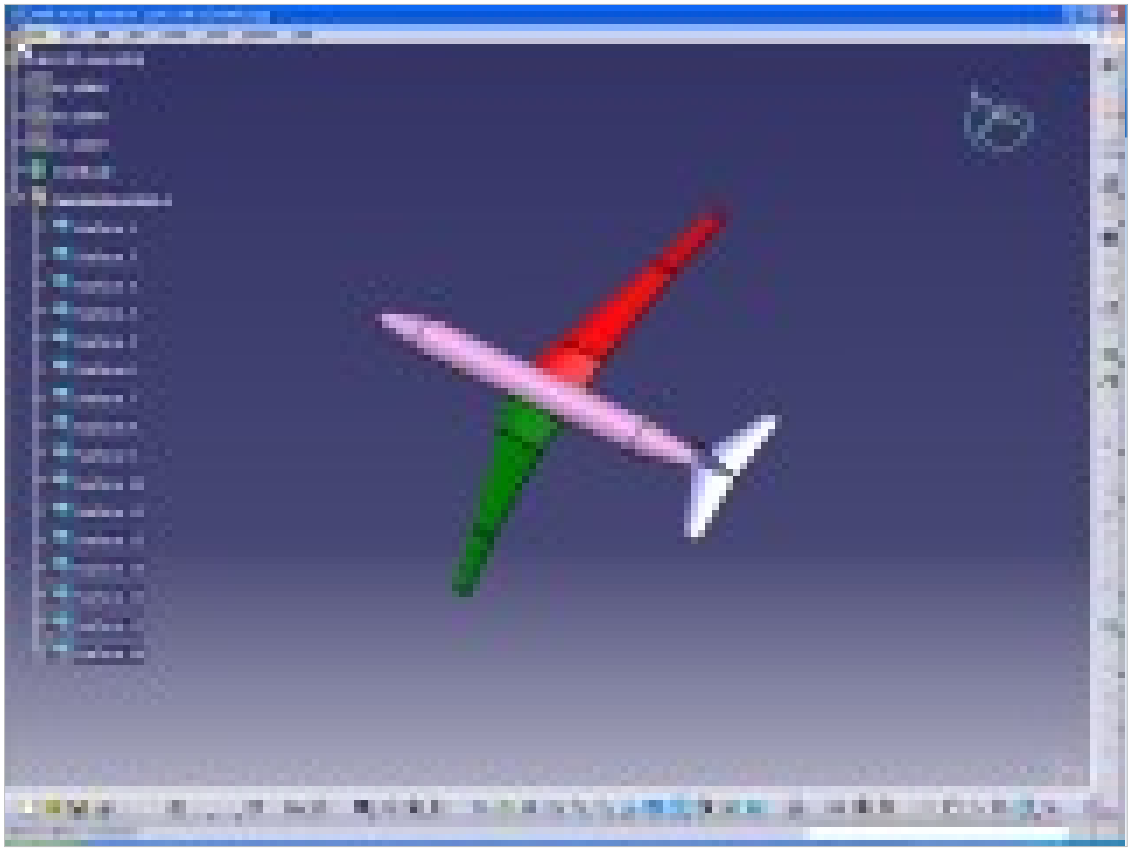
Here is a GDL aircraft model from TU Delft. All components are generated from one generic Airfoil shape function.



An Airship developed as a training exercise, also at the TU Delft.

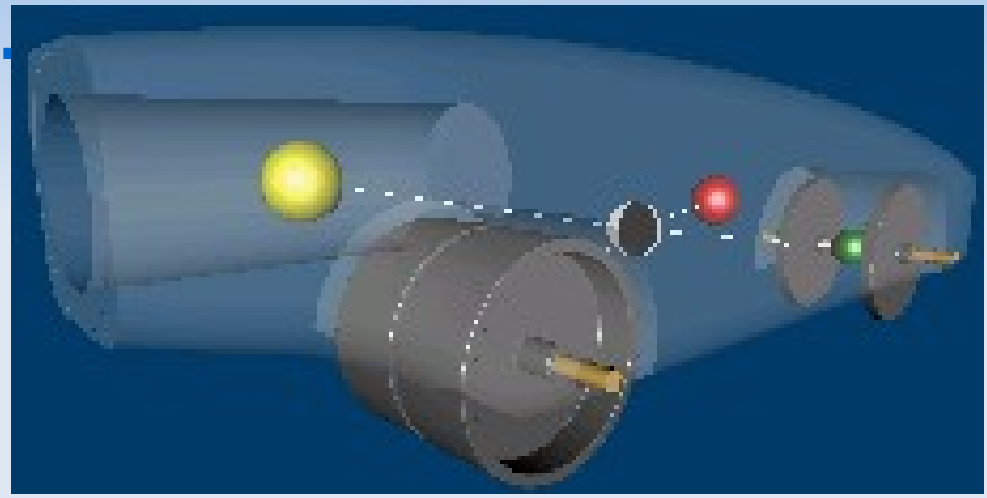


One screen from a User Interface from a electrical pin assignment application, part of an aircraft Wiring Harness design system being developed in partnership between Industry and the TU Delft.



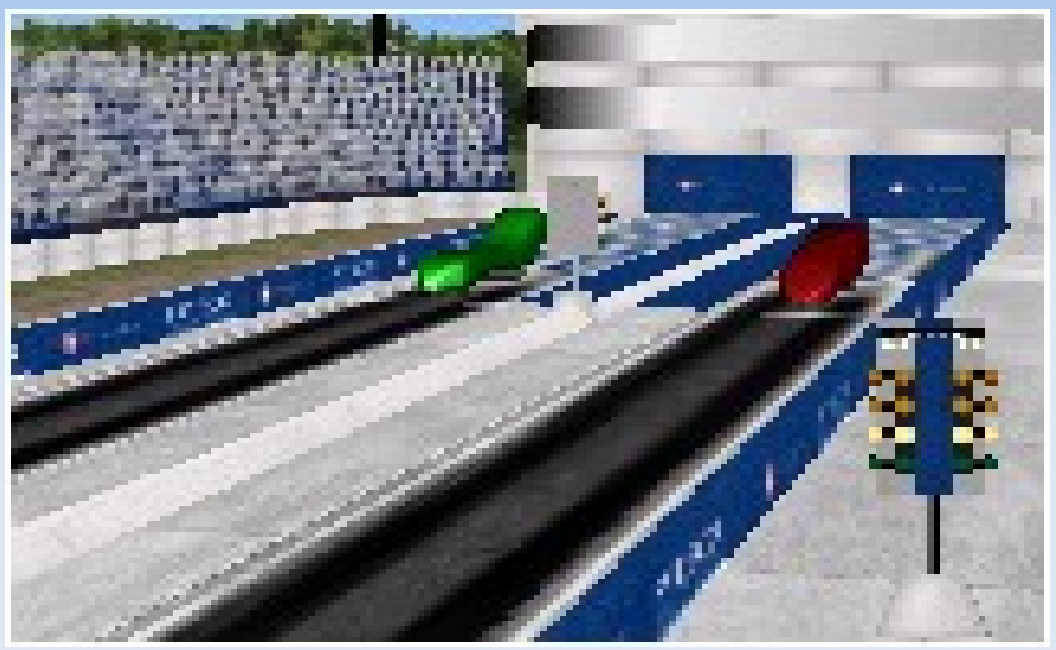
GDL Aircraft model rendered in CATIA.

Sample Outputs



Whitebox CO2 Dragster rendered in a web browser using X3D.

Sample Outputs



Virtual Race for the Dragster, running over the web and making use of X3D with animation.

Conclusion - 21st century KBE

The tools are here today to achieve the Goals:

- Exponential Speedup for developing – provided by Dynamic, Declarative language
- Ease of Deployment – provided by web-centric system
- Longevity of Application Coce – provided by Standards-based core language

22

I would like to conclude by reviewing the 21st century KBE goals that we established, along with how they can be achieved.

First, exponential speedup of development is possible with well-established techniques including the use of a Dynamic, Declarative language. Genworks GDL continues on that tradition.

Second, ease of deployment is possible by using a web-centric system. Genworks GDL makes this a natural approach by extending KBE modeling principles into the realm of modeling web pages and web applications.

Finally, we can achieve maximum longevity of application code by complying as much as possible with Industry Standards. Genworks GDL accomplishes this with the Grandaddy of all Programming Language standards, ANSI CL.

Discussion

- Author's Contact Information
  - Dave Cooper, Product Development Head
  - david.cooper@genworks.com
  - 248 330 2979

Now I believe we have a few minutes for any questions, and discussion.