Final year presentation - David Birch

UNIFYING PROCEDURAL GRAPHICS

(FOR THE GPGPU)

Supervisor: Prof. Duncan F Gillies
Second marker: Dr. Andrew Davison

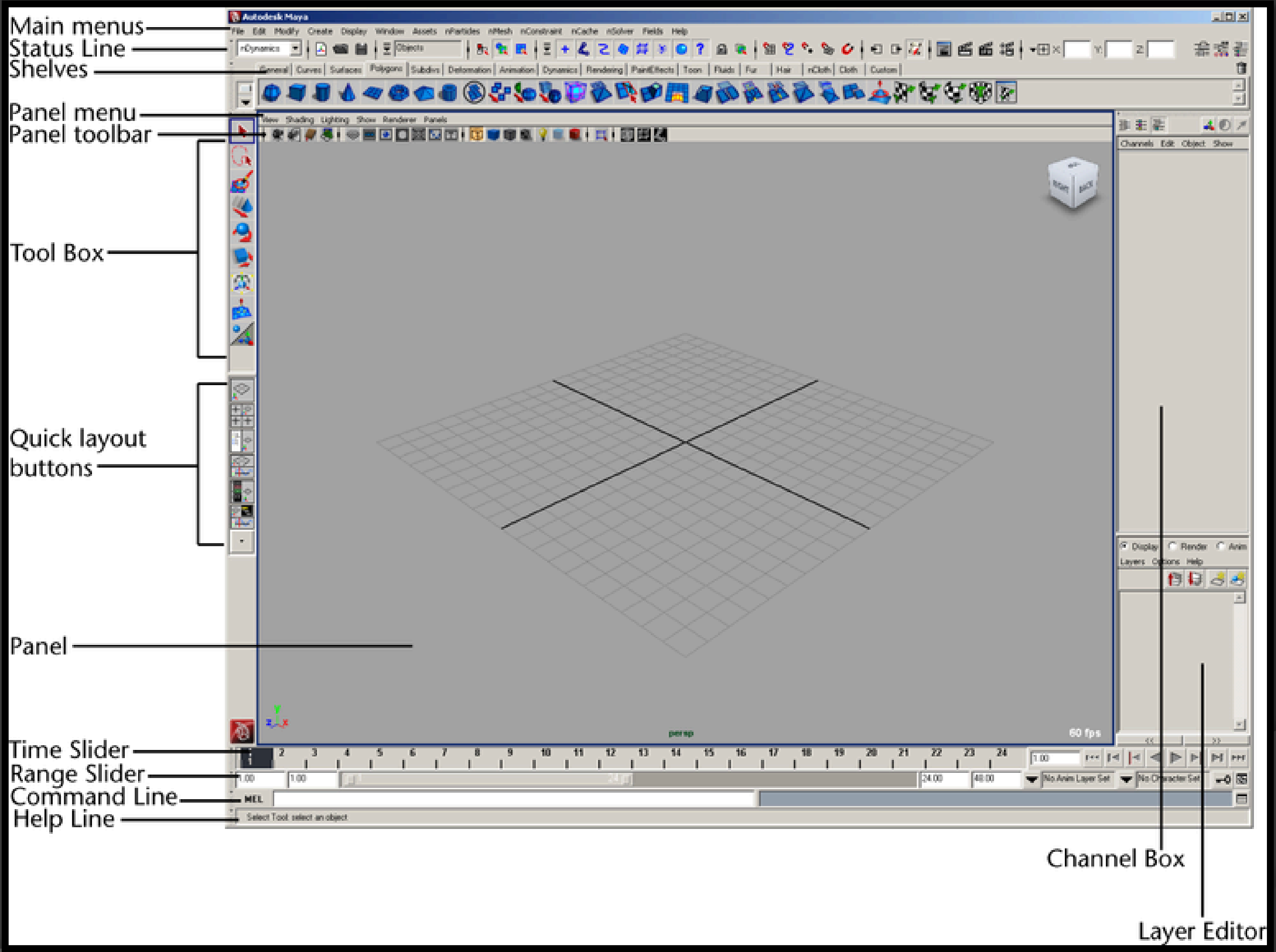
Empire Total War

www.creative-assembly.co.uk

Problem Definition

- Modern graphics scenes are complex requiring huge volumes of content to create compelling scenes.
- This content requirement is increasingly exceeding current creation, storage and delivery mechanisms.

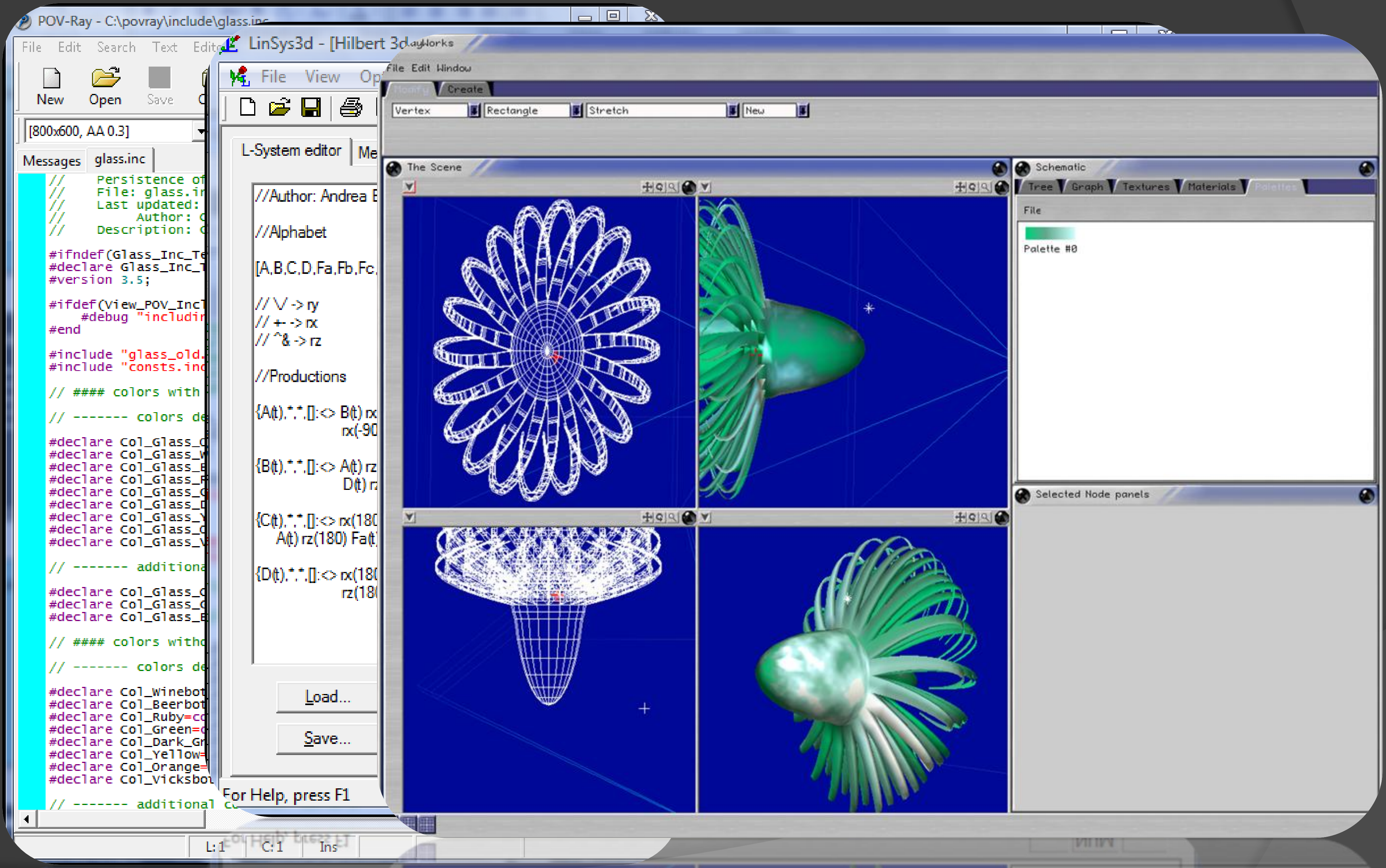
Current Interactive Modellers


Main menus
Status Line
Shelves
Panel menu
Panel toolbar
Tool Box
Quick layout buttons
Panel
Time Slider
Range Slider
Command Line
Help Line
Channel Box
Layer Editor
Autodesk Maya
File Edit Modify Create Display Window Assets nParticles nMesh nConstraint nCache nSolver Fields Help
General Curves Surfaces Polygons Subdivs Deformation Animation Dynamics Rendering PaintEffects Toon Fluids Fur Hair nCloth Cloth Custom
View Shading Lighting Show Renderer Panels
Channels Edit Object Show
Display Render Anim
Layers Options Help
persp
60 fps
MEL
Select Tool: select an object

The Solution

- Algorithmic or Procedural graphics:
  - Complex Models
  - Similar Models
  - Small Storage Requirements
  - On demand generation
  - Reuse
  - Non-linear editing

Current Procedural Modellers



```
//     Persistence of
//     File: glass.in
//     Last updated:
//         Author: C
//     Description: C

#ifndef(Glass_Inc_Te
#declare Glass_Inc_T
#version 3.5;

#ifdef(View_POV_IncT
     #debug "includin
#end

#include "glass_old.
#include "consts.inc

// #### colors with

// ------- colors de

#declare Col_Glass_C
#declare Col_Glass_W
#declare Col_Glass_E
#declare Col_Glass_F
#declare Col_Glass_G
#declare Col_Glass_D
#declare Col_Glass_Y
#declare Col_Glass_C
#declare Col_Glass_V

// ------- additiona

#declare Col_Glass_C
#declare Col_Glass_C
#declare Col_Glass_E

// #### colors witho

// ------- colors de

#declare Col_Winebot
#declare Col_Beerbot
#declare Col_Ruby=co
#declare Col_Green=c
#declare Col_Dark_Gr
#declare Col_Yellow=
#declare Col_Orange=
#declare Col_Vicksbot

// ------- additional co
```

```
//Author: Andrea E

//Alphabet

[A,B,C,D,Fa,Fb,Fc,

// \/ -> ry
// +- -> rx
// ^& -> rz

//Productions

{A(t),*,*,[]:<> B(t) rx
              rx(-90

{B(t),*,*,[]:<> A(t) rz
              D(t) rz

{C(t),*,*,[]:<> rx(180
     A(t) rz(180) Fa(t)

{D(t),*,*,[]:<> rx(180
              rz(180
```

But…

- Complexity
- Scripting languages
- Skills mismatch
- & Fragmentation

The Solution:

UNIFICATION

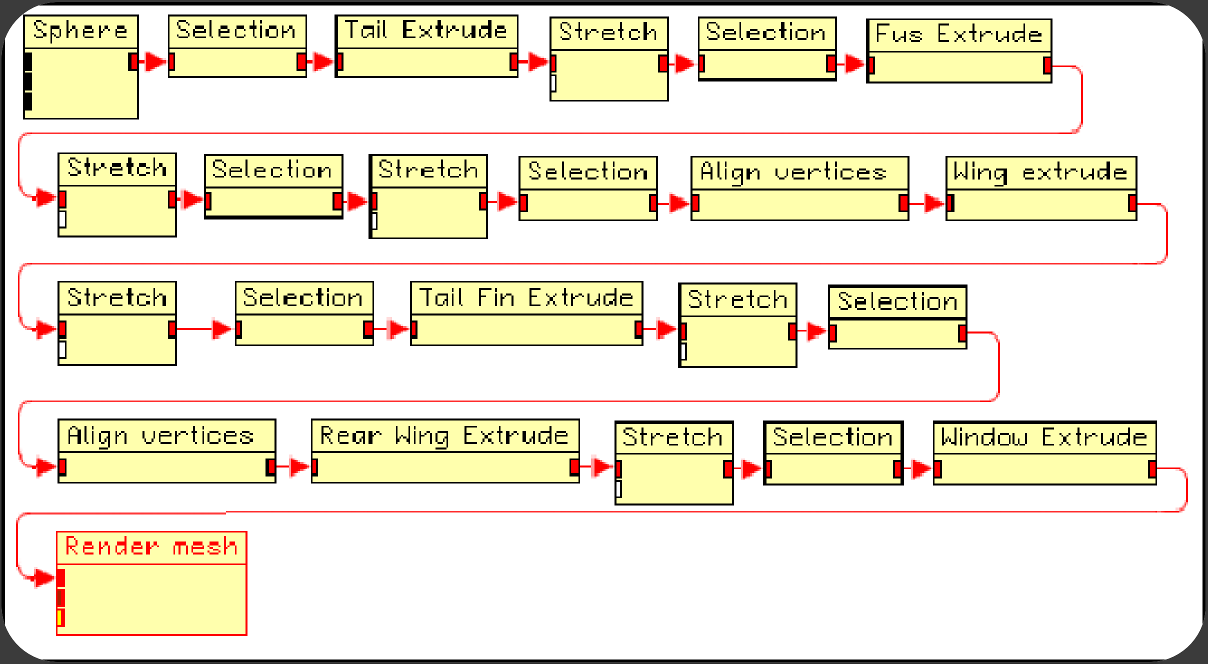
&

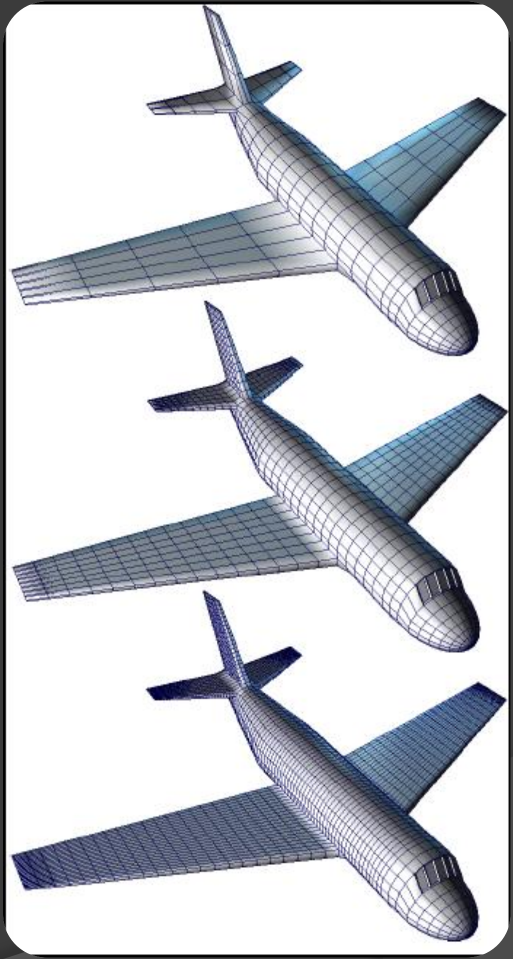
SIMPLIFICATION

ClayWorks:

A System for the Non-Linear Modelling of Deformable Procedural Shapes

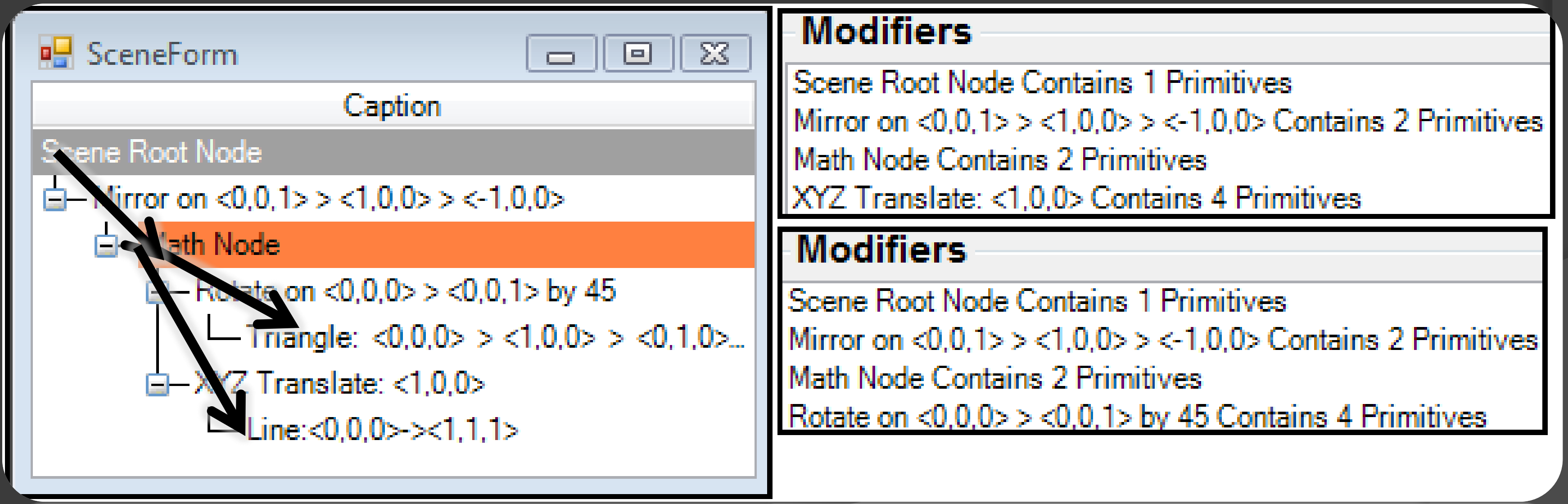
T. Lewis and M. W. Jones





Our Approach

- Tree based pipelines
- Leaf nodes are graphical primitives
- One pipeline for each group of primitives.
- Pipelines flow from the root node down to the primitives.



Scene Root Node
- Mirror on <0,0,1> > <1,0,0> > <-1,0,0>
  - Math Node
    - Rotate on <0,0,0> > <0,0,1> by 45
      - Triangle: <0,0,0> > <1,0,0> > <0,1,0>...
    - XYZ Translate: <1,0,0>
      - Line:<0,0,0>-><1,1,1>

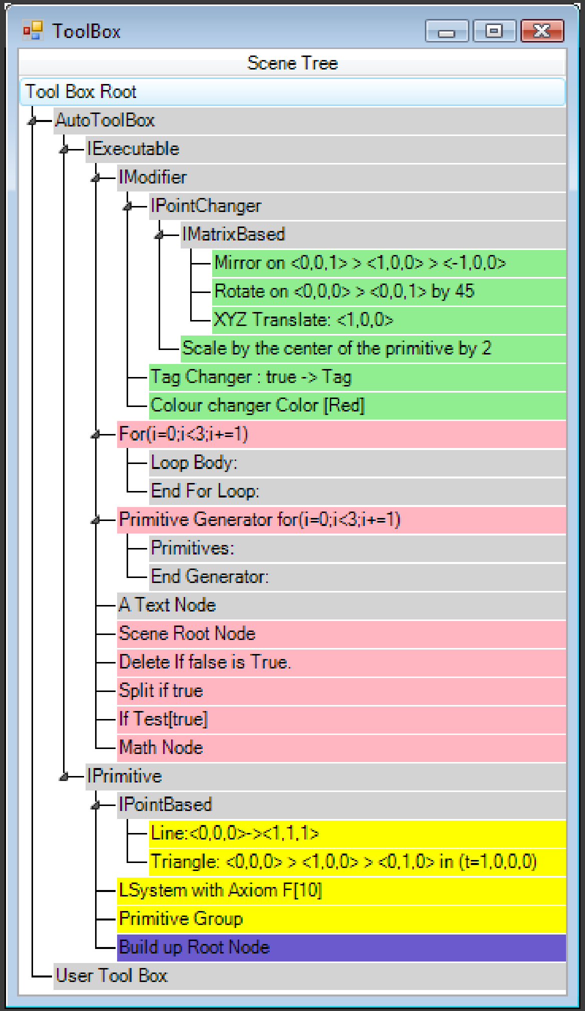
Modifiers

Scene Root Node Contains 1 Primitives
Mirror on <0,0,1> > <1,0,0> > <-1,0,0> Contains 2 Primitives
Math Node Contains 2 Primitives
XYZ Translate: <1,0,0> Contains 4 Primitives

Modifiers

Scene Root Node Contains 1 Primitives
Mirror on <0,0,1> > <1,0,0> > <-1,0,0> Contains 2 Primitives
Math Node Contains 2 Primitives
Rotate on <0,0,0> > <0,0,1> by 45 Contains 4 Primitives

Language

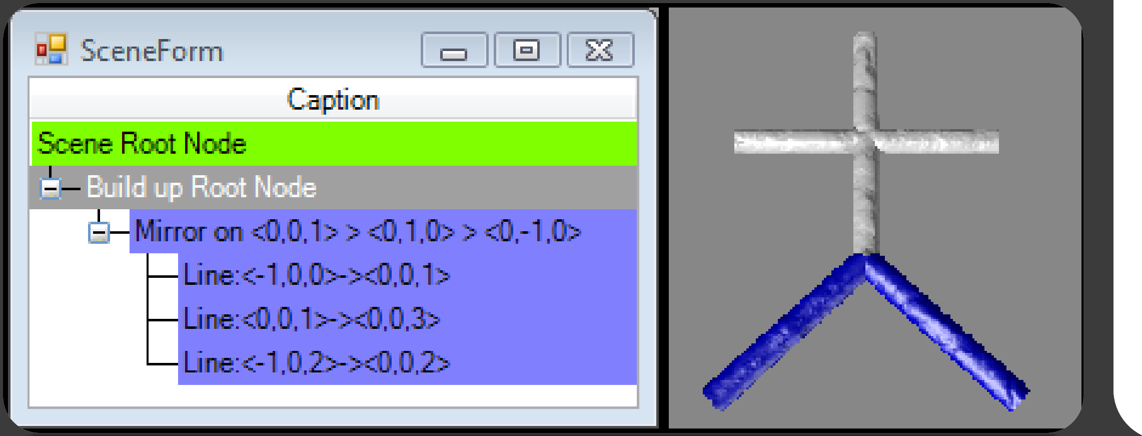


```
ToolBox
Scene Tree
Tool Box Root
  AutoToolBox
    IExecutable
      IModifier
        IPointChanger
          IMatrixBased
            Mirror on <0,0,1> > <1,0,0> > <-1,0,0>
            Rotate on <0,0,0> > <0,0,1> by 45
            XYZ Translate: <1,0,0>
          Scale by the center of the primitive by 2
        Tag Changer : true -> Tag
        Colour changer Color [Red]
      For(i=0;i<3;i+=1)
        Loop Body:
        End For Loop:
      Primitive Generator for(i=0;i<3;i+=1)
        Primitives:
        End Generator:
      A Text Node
      Scene Root Node
      Delete If false is True.
      Split if true
      If Test[true]
      Math Node
    IPrimitive
      IPointBased
        Line:<0,0,0>-><1,1,1>
        Triangle: <0,0,0> > <1,0,0> > <0,1,0> in (t=1,0,0,0)
      LSystem with Axiom F[10]
      Primitive Group
      Build up Root Node
  User Tool Box
```

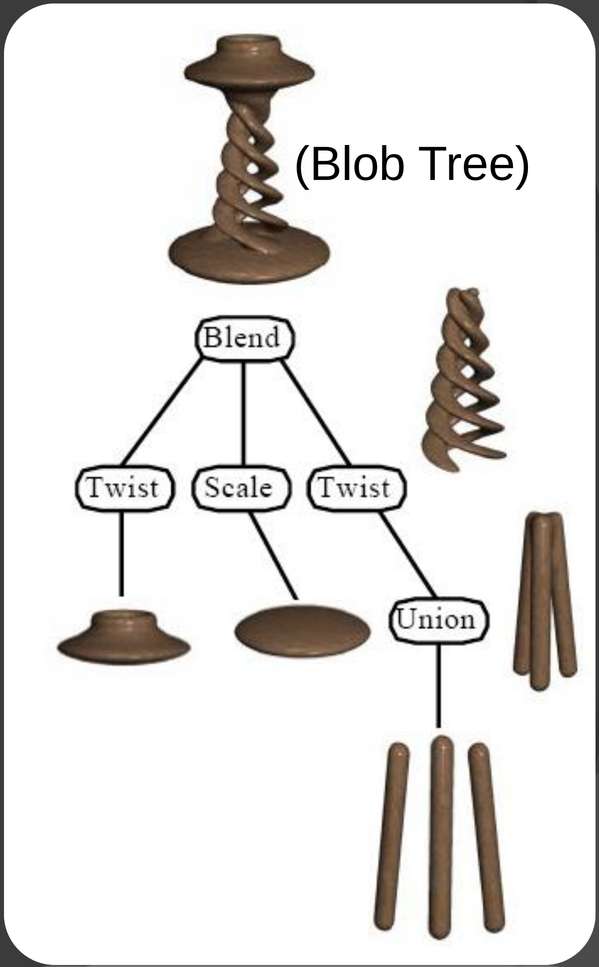
- Primitives:
  - Lines & Triangles
  - Cuboids, Spheres
  - Planes
- Modifiers:
  - Translate, Rotate, Scale
  - Mirror & Colour Changer
- Other pipeline modifications such as:
  - Truncation
  - Repetition
  - & Primitive filtering

Unified: Buildup Semantics

- Tree-based
  Pipelined execution
- CSG like “Buildup”
  Tree semantics



Scene Root Node
- Build up Root Node
  - Mirror on <0,0,1> > <0,1,0> > <0,-1,0>
    - Line:<-1,0,0>-><0,0,1>
    - Line:<0,0,1>-><0,0,3>
    - Line:<-1,0,2>-><0,0,2>

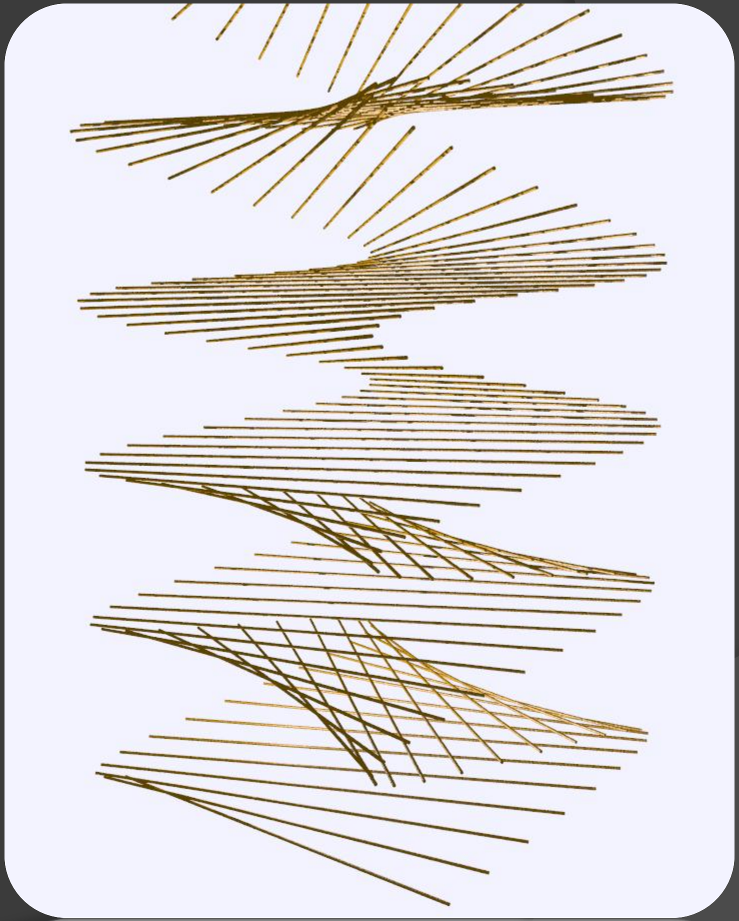


Unified: Stream and Batched Execution

- Two methods of executing modifiers:
- Stream based execution independent execution on each primitive
- Batched execution – all primitives processed by one node before being passed to the next.

Unified: Mathematical Scripting

- Almost all attributes in the scene tree are actually mathematical expressions.
- Allowing mathematical modelling:



Unified: Imperative Constructs

- We provide the following features to our language:
  - If tests – to truncate a pipeline
  - For loops to repeat a given segment of pipeline
  - Splitter nodes which allow sharing of primitives
  - Filter nodes which selectively remove primitives
- We also allow mathematical variables to flow through the tree

Unified: LSystems

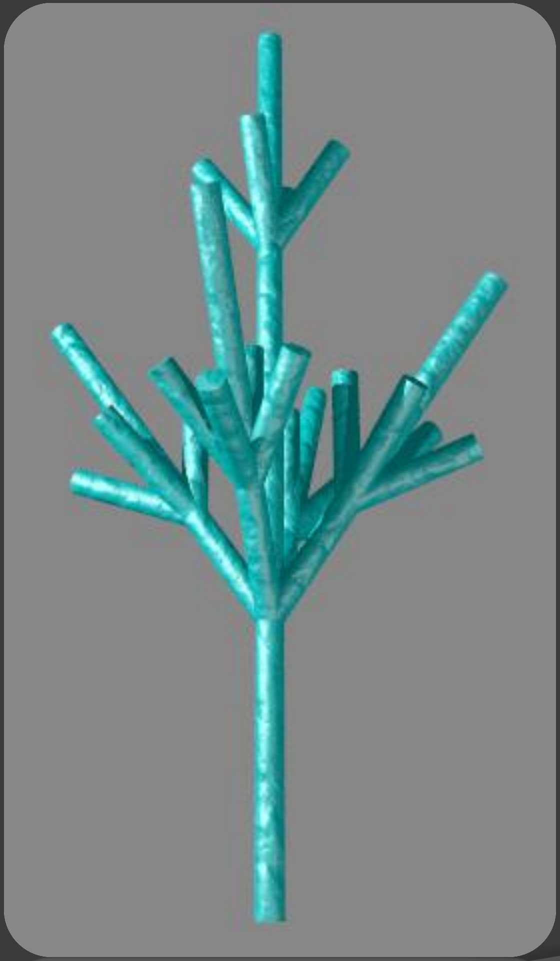
- We implement a Bracketed Parameterised LSystem with mathematical expressions.
- An LSystem is a production system for commands for a drawing robot (the “turtle”).

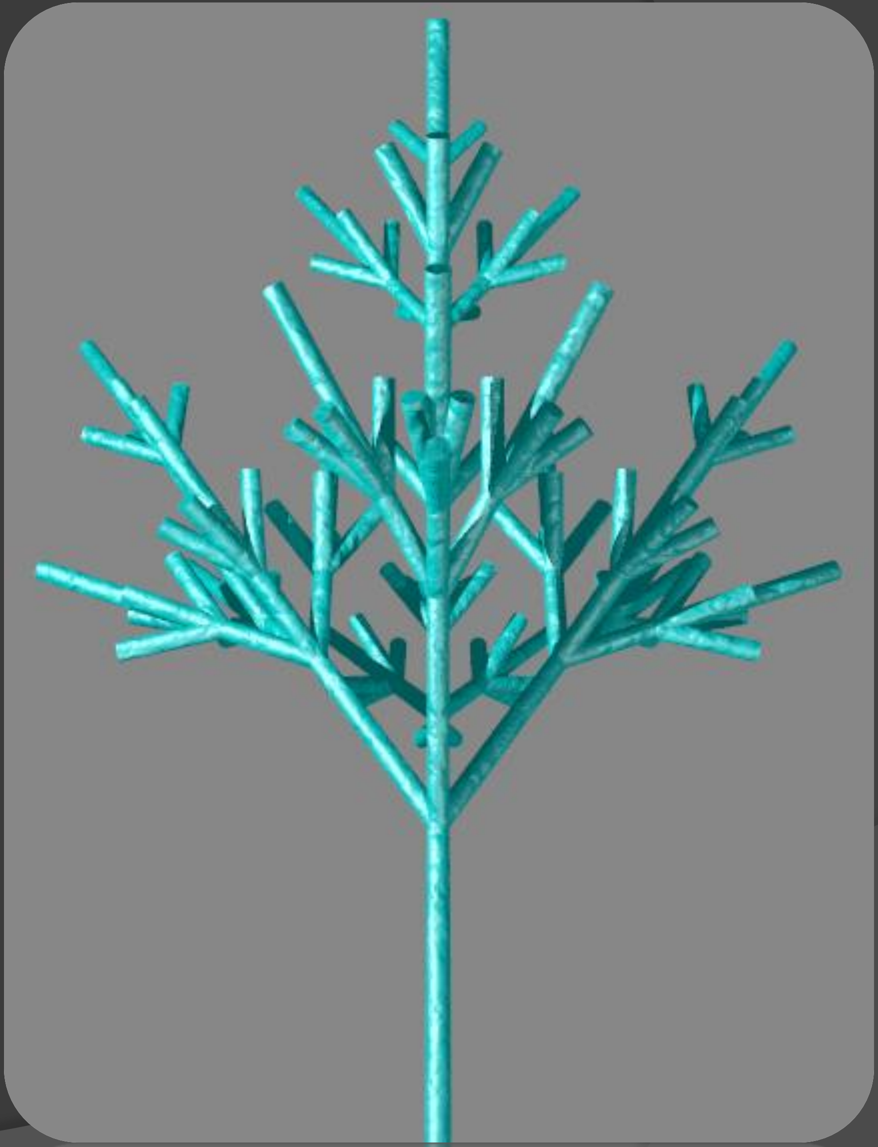
- Commands: F, X, Y, Z, +, -, {, }
- Axiom: “X[45]F[10]”
- Productions: “F[10] -> F[10]X[45]F[5]”
- Giving: “X[45]**F[10]X[45]F[5]**”

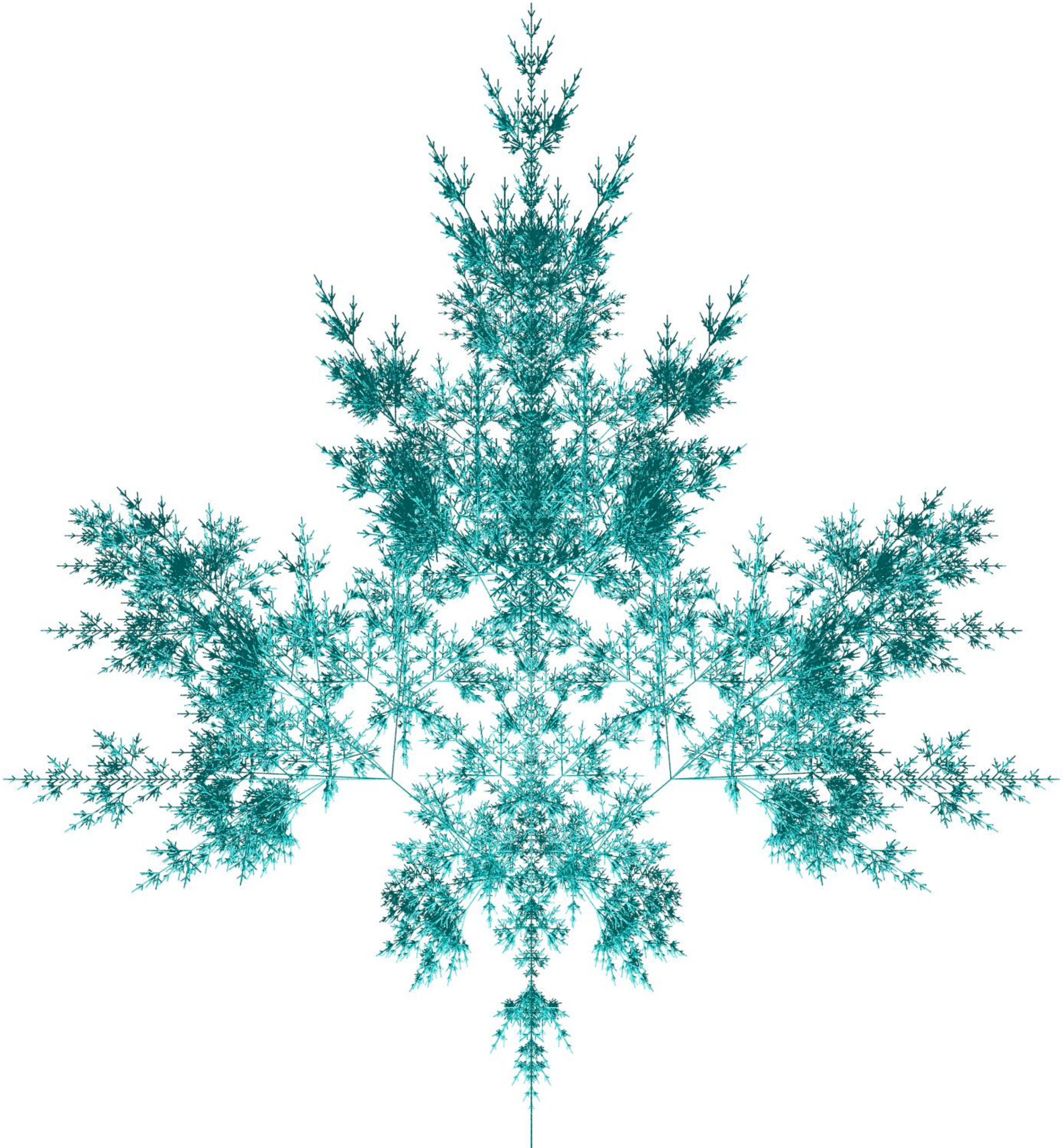
- Actually:

  “F[d] -> F[d*2]X[45]F[Max(d*10,Exp(5))]”

LSystems:







Development Environment:

- Do/undo/redo with full history
- Save/load
- Log system
- Custom highlighting

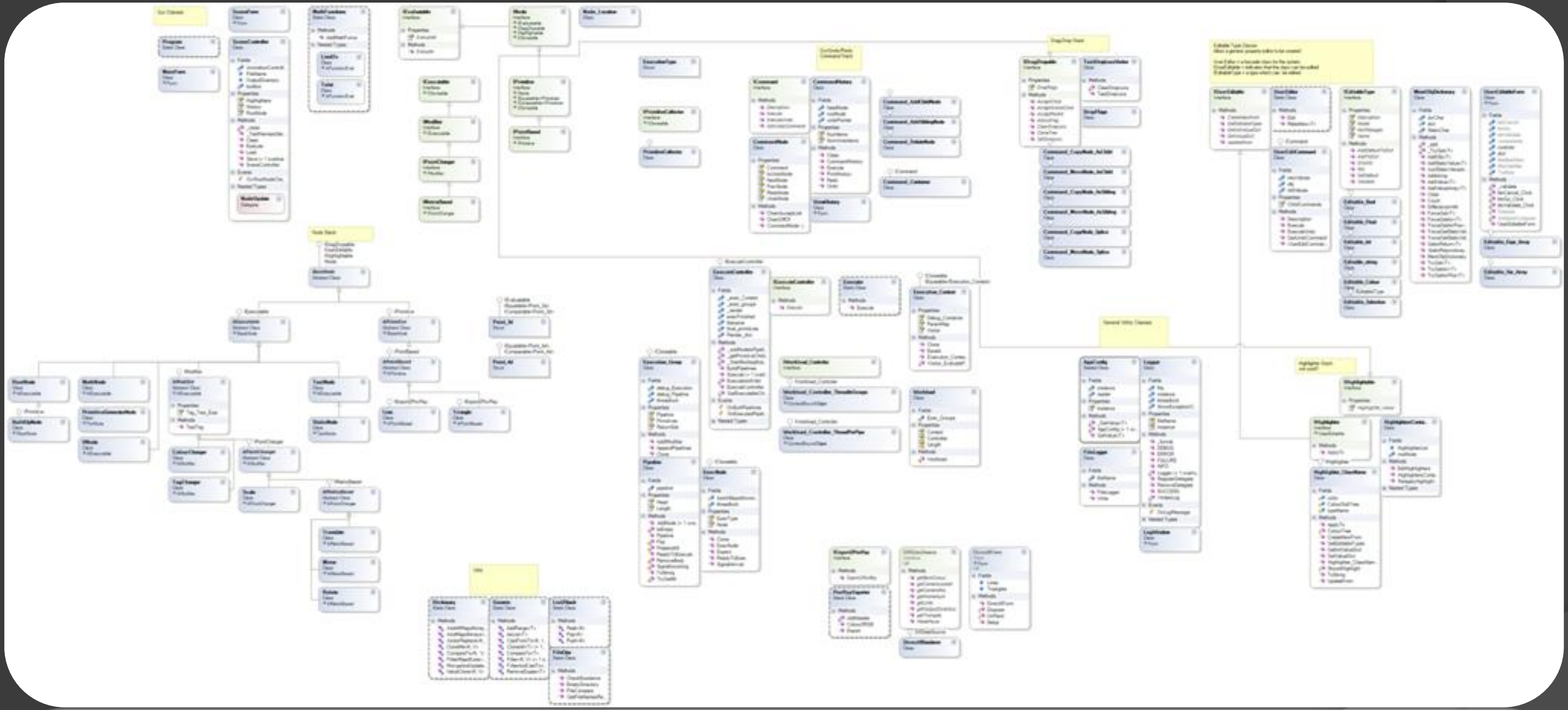
Development Environment:

- Graphically manipulated language
- Typesafe drag and drop
- Consistent auto-generated edit system with validation and help messages
- Visual debugging

Demo!

- Simple example
  - Show interface
  - Show pipelines
  - Show workflow

Codebase: C# 3.5 & Visual Studio



22,082 lines in 167+ classes

Pipeline Creation

```
BuildPipelines(IModifier rootNode) {
    // set up data
    List<Pipeline> pipelines = new List<Pipeline>();
    List<IModifier> branches = GetExecutableChildren(rootNode);
    List<Primitive> prims = GetPrimitiveChildren(rootNode);

    if (prims.Count>0) { // find primitives and start a pipeline
         pipelines.Add( new Pipeline(prims));
    }

    foreach (IModifier modifier in branches) { // recurse
                  pipelines.AddAll( BuildPipelines(modifier));
    }

    foreach (Pipeline pipe in pipelines) { // extend pipelines
                  pipe.InsertStage(rootNode);
    }

    return pipelines;
}
```

Pipeline Execution

- The list of pipelines to execute is split upon their first modifier.
- The number of modifier which all the pipelines in each group share is found.
- Each such pipeline section is then sent to a ThreadPool for multi-threaded execution.
- On completion of a section the remaining pipelines are returned and split as above.
- Care is taken of Batch modifiers and the signalling they require.

NCalc Expression Evaluator

- Extensible open source C# expression evaluator library
- Multiple data types, delegate extensible function list & events to evaluate parameters & functions
- A mathematical context (variable to value mapping) flows through the pipeline

CodePlex
Open Source Project Community

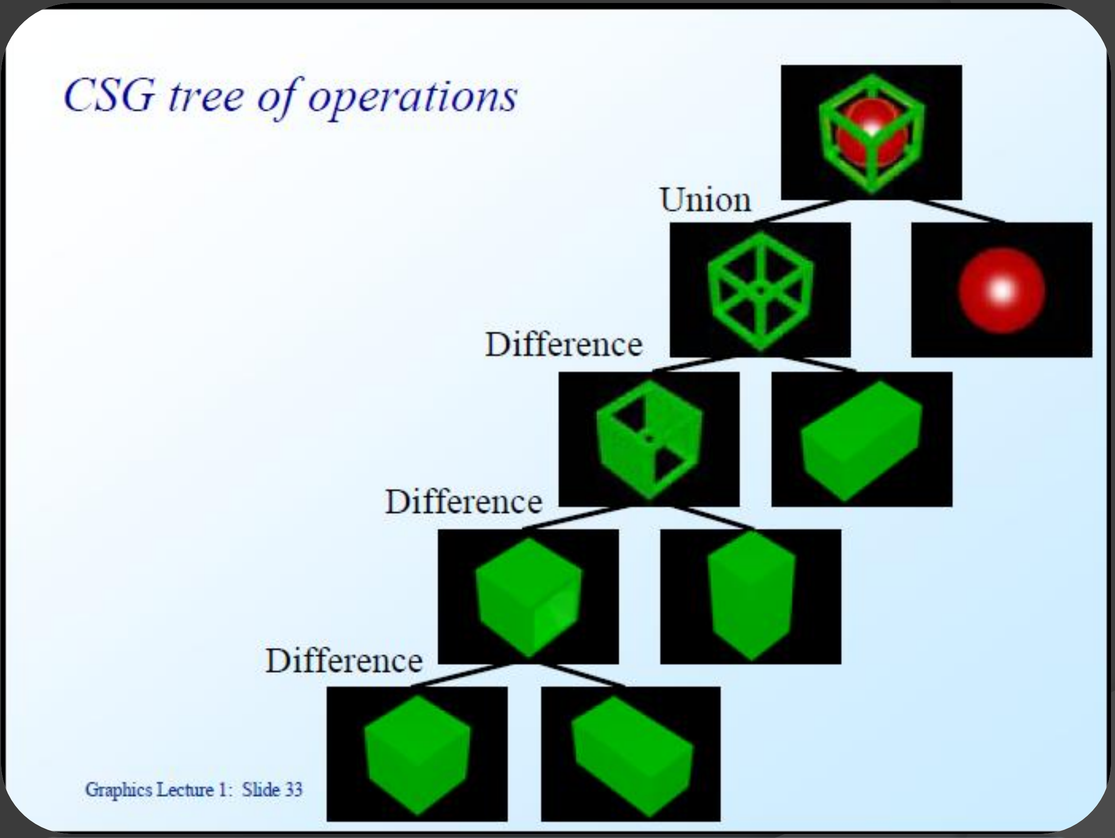
evaluant

Further Work (Simplicity)

- Interactive modelling!
- Integrate tools such as translate and scale into the DirectX renderer
- Automatic extension of the scene tree.
- Allow primitive drawing in DirectX renderer
- Methods of selecting primitives

Further Work (Unity)

- Move to 3d primitives & modifiers such extrude.
- This allows Constructive Solid Geometry (CSG)

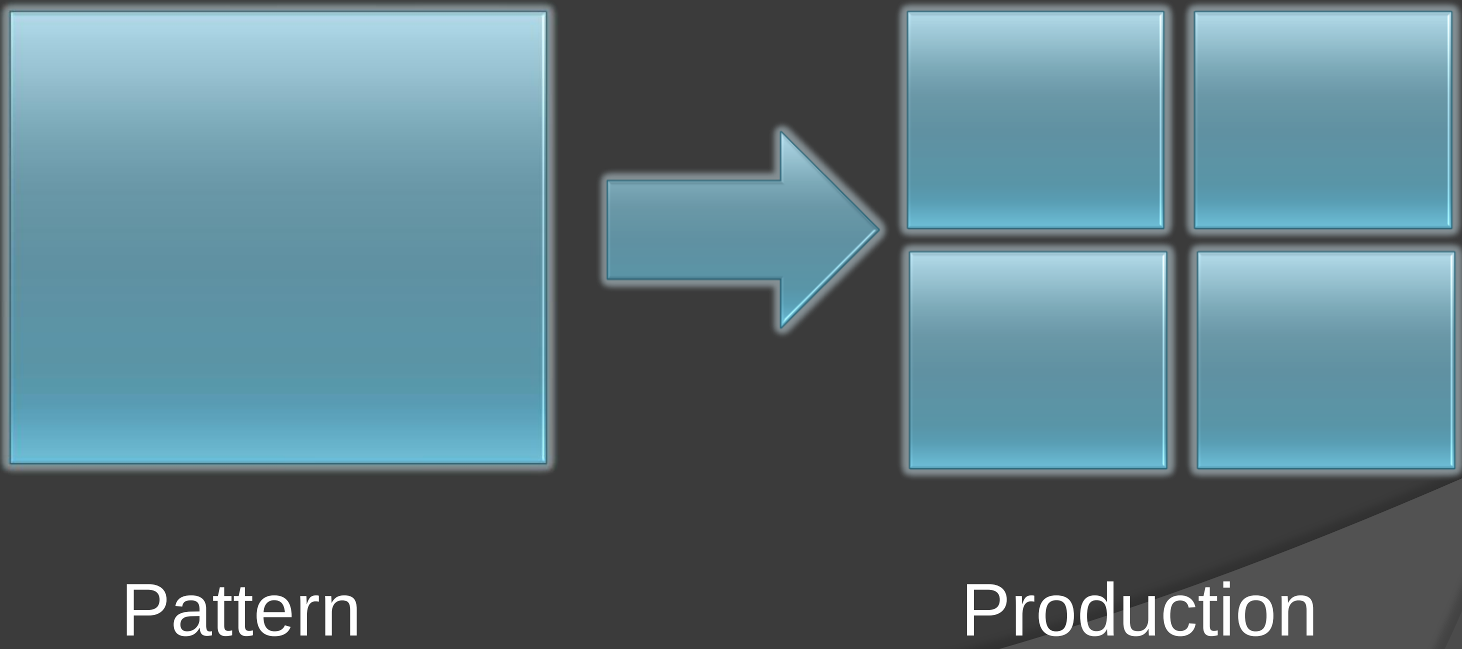


CSG tree of operations

Graphics Lecture 1: Slide 33

Further Work (Unity)

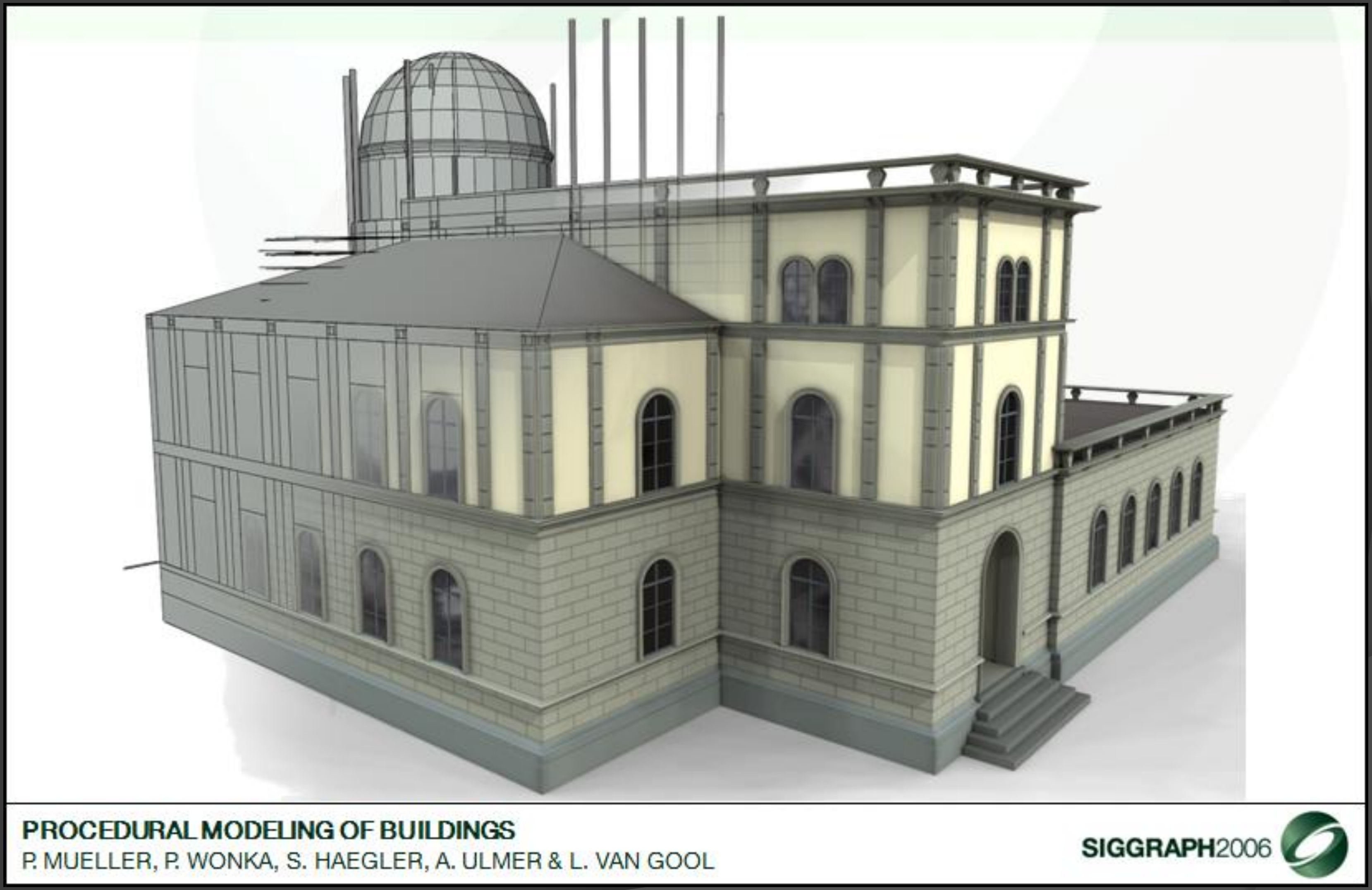
- Shape Grammar
- An LSystem but with graphical primitives



Pattern

Production

Example: City Engine

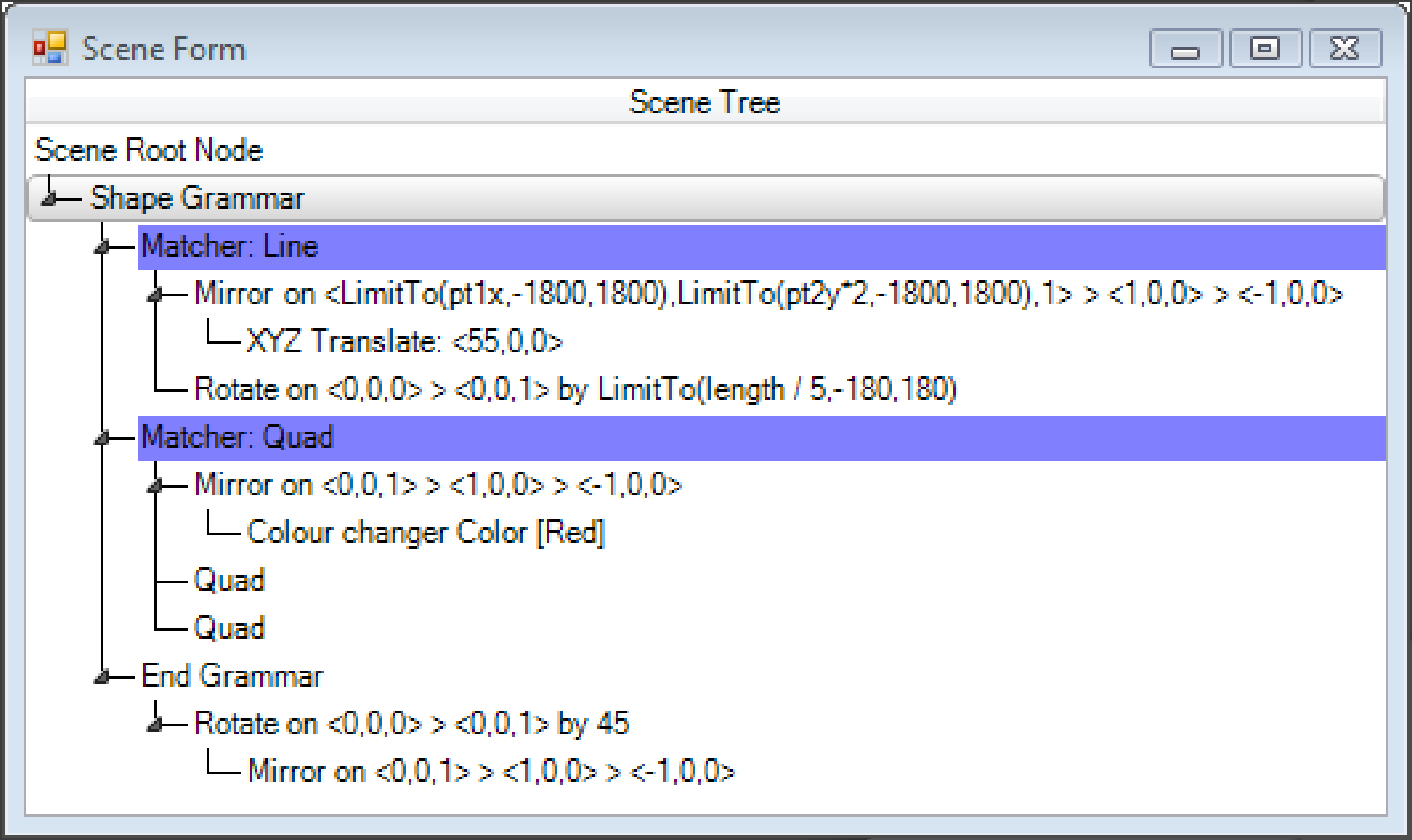


PROCEDURAL MODELING OF BUILDINGS

P. MUELLER, P. WONKA, S. HAEGLER, A. ULMER & L. VAN GOOL

SIGGRAPH2006

Integration:



```
Scene Root Node
└─ Shape Grammar
    ├─ Matcher: Line
    │   ├─ Mirror on <LimitTo(pt1x,-1800,1800),LimitTo(pt2y*2,-1800,1800),1> > <1,0,0> > <-1,0,0>
    │   │   └─ XYZ Translate: <55,0,0>
    │   └─ Rotate on <0,0,0> > <0,0,1> by LimitTo(length / 5,-180,180)
    ├─ Matcher: Quad
    │   ├─ Mirror on <0,0,1> > <1,0,0> > <-1,0,0>
    │   │   └─ Colour changer Color [Red]
    │   ├─ Quad
    │   └─ Quad
    └─ End Grammar
        └─ Rotate on <0,0,0> > <0,0,1> by 45
            └─ Mirror on <0,0,1> > <1,0,0> > <-1,0,0>
```

Further Work (Performance)

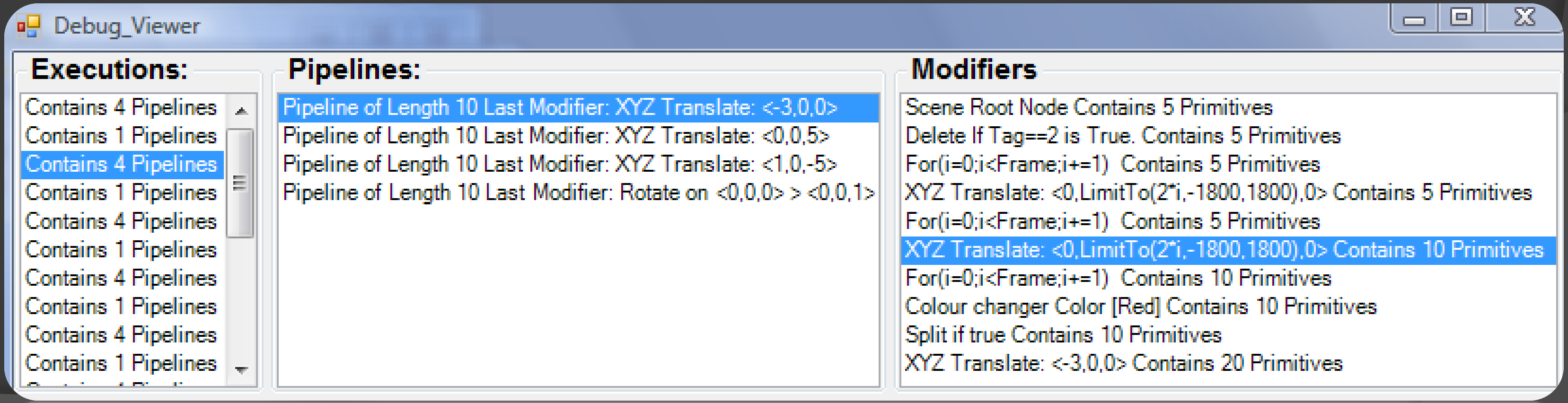
- Aggressive Threading:
  - Multiple threads per pipeline section
  - Work Splitting Algorithms for modifiers with large workloads (10,000 primitives +)
  - Intelligent algorithms required!

Further Work (Performance)

- Cuda – a C extension which runs on NVidia Tesla graphics cards, providing general purpose computing with 100x throughput of modern CPU’s
- 240 “cores” support up to 30,000 running threads.

Further Work (Performance)

- Generate Meta Data of pipelines.
- Translate each modifier to C/Cuda code
- Aggressively threaded – one thread per primitive
- Execute multiple pipeline sections on the graphics



Executions:

```
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
```

Pipelines:

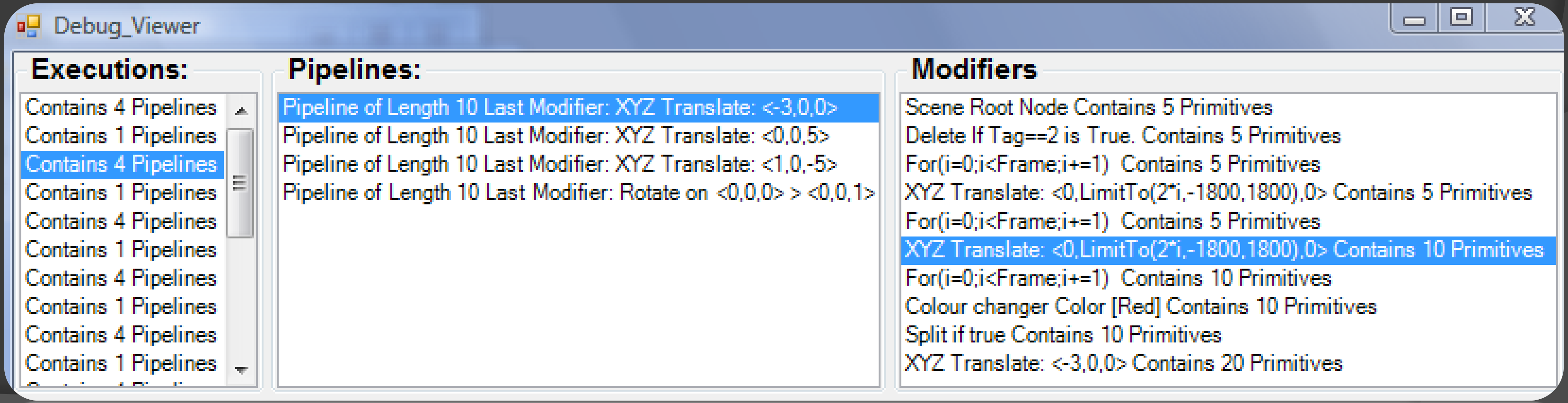
```
Pipeline of Length 10 Last Modifier: XYZ Translate: <-3,0,0>
Pipeline of Length 10 Last Modifier: XYZ Translate: <0,0,5>
Pipeline of Length 10 Last Modifier: XYZ Translate: <1,0,-5>
Pipeline of Length 10 Last Modifier: Rotate on <0,0,0> > <0,0,1>
```

Modifiers

```
Scene Root Node Contains 5 Primitives
Delete If Tag==2 is True. Contains 5 Primitives
For(i=0;i<Frame;i+=1)  Contains 5 Primitives
XYZ Translate: <0,LimitTo(2*i,-1800,1800),0> Contains 5 Primitives
For(i=0;i<Frame;i+=1)  Contains 5 Primitives
XYZ Translate: <0,LimitTo(2*i,-1800,1800),0> Contains 10 Primitives
For(i=0;i<Frame;i+=1)  Contains 10 Primitives
Colour changer Color [Red] Contains 10 Primitives
Split if true Contains 10 Primitives
XYZ Translate: <-3,0,0> Contains 20 Primitives
```

Further Work (Performance)

- **Active Semantic Caching:**
- Cache executions along with the pipelines that generated them.
- When a new execution is required and a similar execution is cached we can compute the extra stages and not the whole pipeline



Executions:

```
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
Contains 4 Pipelines
Contains 1 Pipelines
```

Pipelines:

```
Pipeline of Length 10 Last Modifier: XYZ Translate: <-3,0,0>
Pipeline of Length 10 Last Modifier: XYZ Translate: <0,0,5>
Pipeline of Length 10 Last Modifier: XYZ Translate: <1,0,-5>
Pipeline of Length 10 Last Modifier: Rotate on <0,0,0> > <0,0,1>
```

Modifiers

```
Scene Root Node Contains 5 Primitives
Delete If Tag==2 is True. Contains 5 Primitives
For(i=0;i<Frame;i+=1)  Contains 5 Primitives
XYZ Translate: <0,LimitTo(2*i,-1800,1800),0> Contains 5 Primitives
For(i=0;i<Frame;i+=1)  Contains 5 Primitives
XYZ Translate: <0,LimitTo(2*i,-1800,1800),0> Contains 10 Primitives
For(i=0;i<Frame;i+=1)  Contains 10 Primitives
Colour changer Color [Red] Contains 10 Primitives
Split if true Contains 10 Primitives
XYZ Translate: <-3,0,0> Contains 20 Primitives
```

Demos & Questions?

- Demos:
  - Clock
  - Helix
  - Orchard
  - Marching Column
- Extra material:
  - Composite Nodes
  - Mathematical Scripting
  - Selection Channels
  - Primitive Tagging
  - Software Engineering
  - Reflections on C# & .Net 3.5

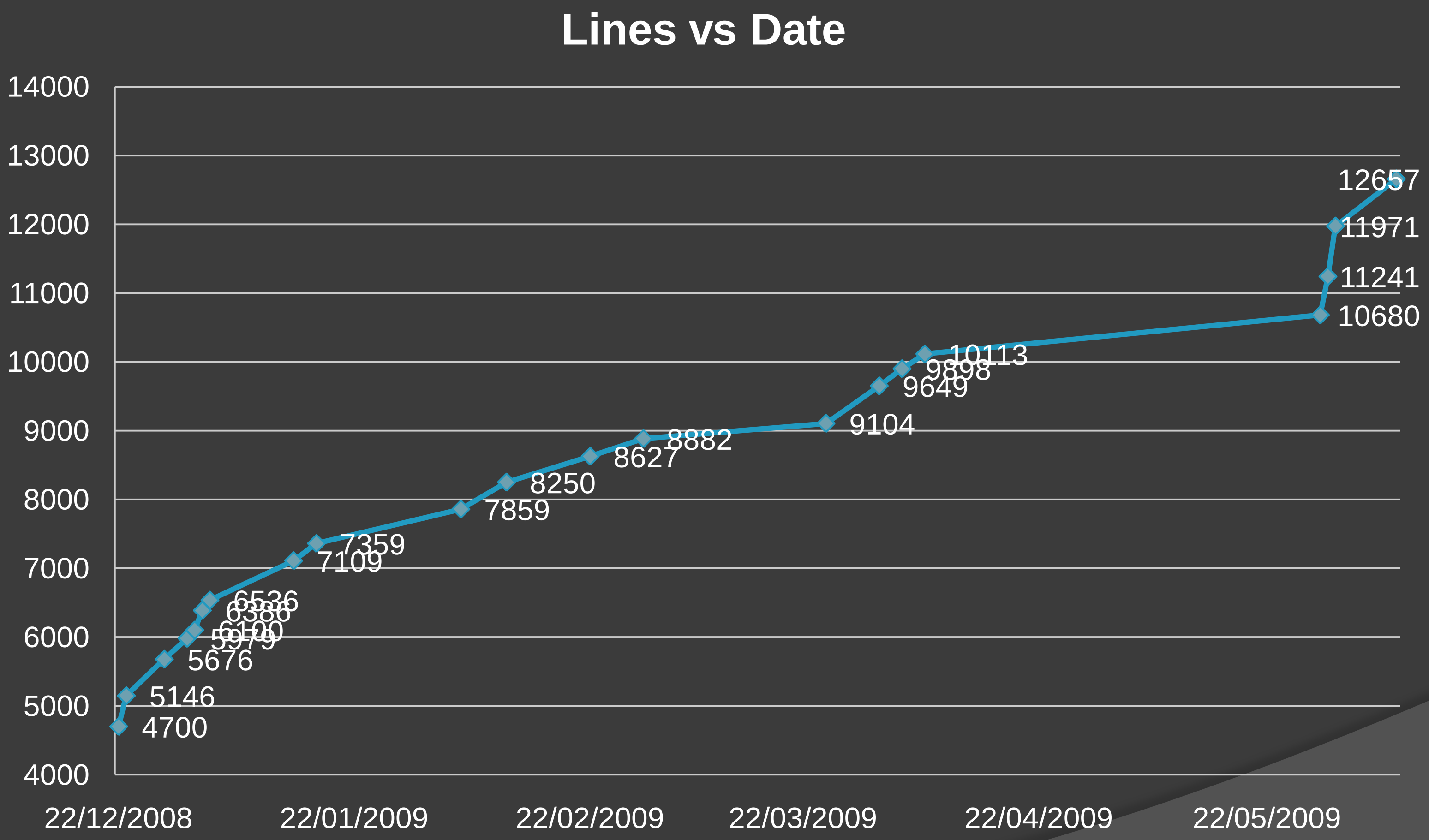
Extra Material - Tagging

- Every primitive has a Tag attribute
- Every modifier has a Tag Test which dictates whether or not to apply the modifier to a given primitive.
- All Tags and Tests are math expressions
- This allows semantic groupings of primitives.
- There are Tag Changer nodes and Filter on Tag modifiers to facilitate this

Extra Material – Selection Channels (ClayWorks)

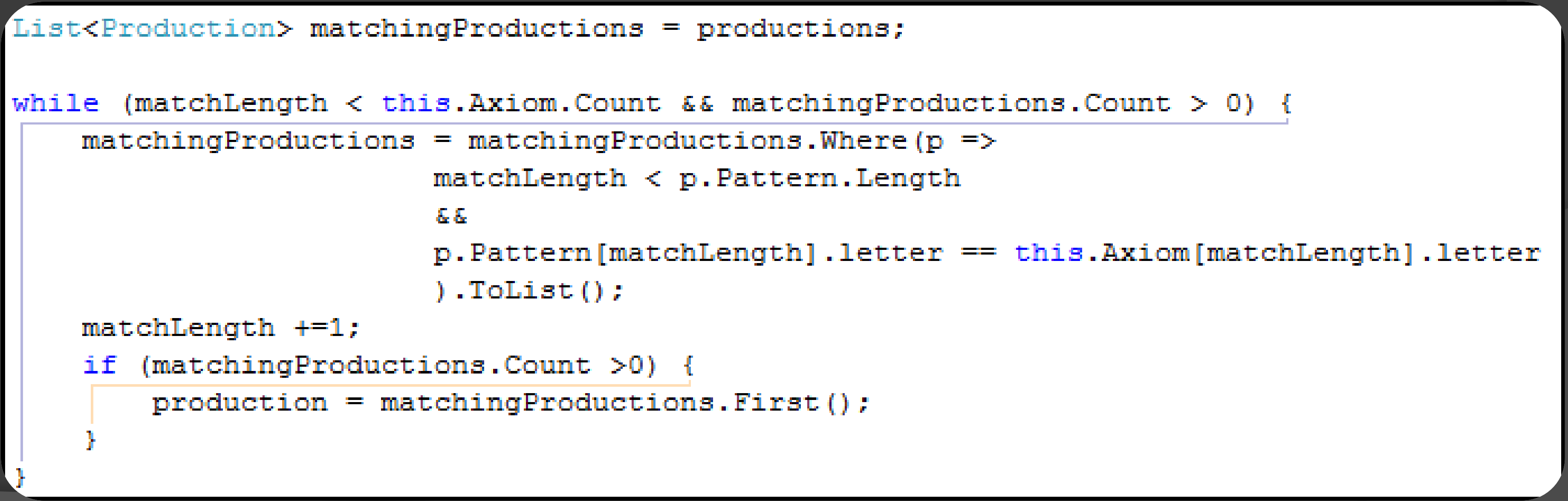
- Selection is made volumetrically via set operators on a number of convex hulls.
- Sphere radius 5 on <0,0,0> UNION Sphere radius 5 on <10,50,5>
- The convex hulls are also passed through the pipelines and are acted upon.
- This avoids brittle selection which is broken when a user modifiers an earlier pipeline stage.

Software Engineering



C# & LINQ

- Introduces Functional constructs into an imperative language:



```
List<Production> matchingProductions = productions;

while (matchLength < this.Axiom.Count && matchingProductions.Count > 0) {
    matchingProductions = matchingProductions.Where(p =>
                          matchLength < p.Pattern.Length
                          &&
                          p.Pattern[matchLength].letter == this.Axiom[matchLength].letter
                          ).ToList();
    matchLength +=1;
    if (matchingProductions.Count >0) {
        production = matchingProductions.First();
    }
}
```