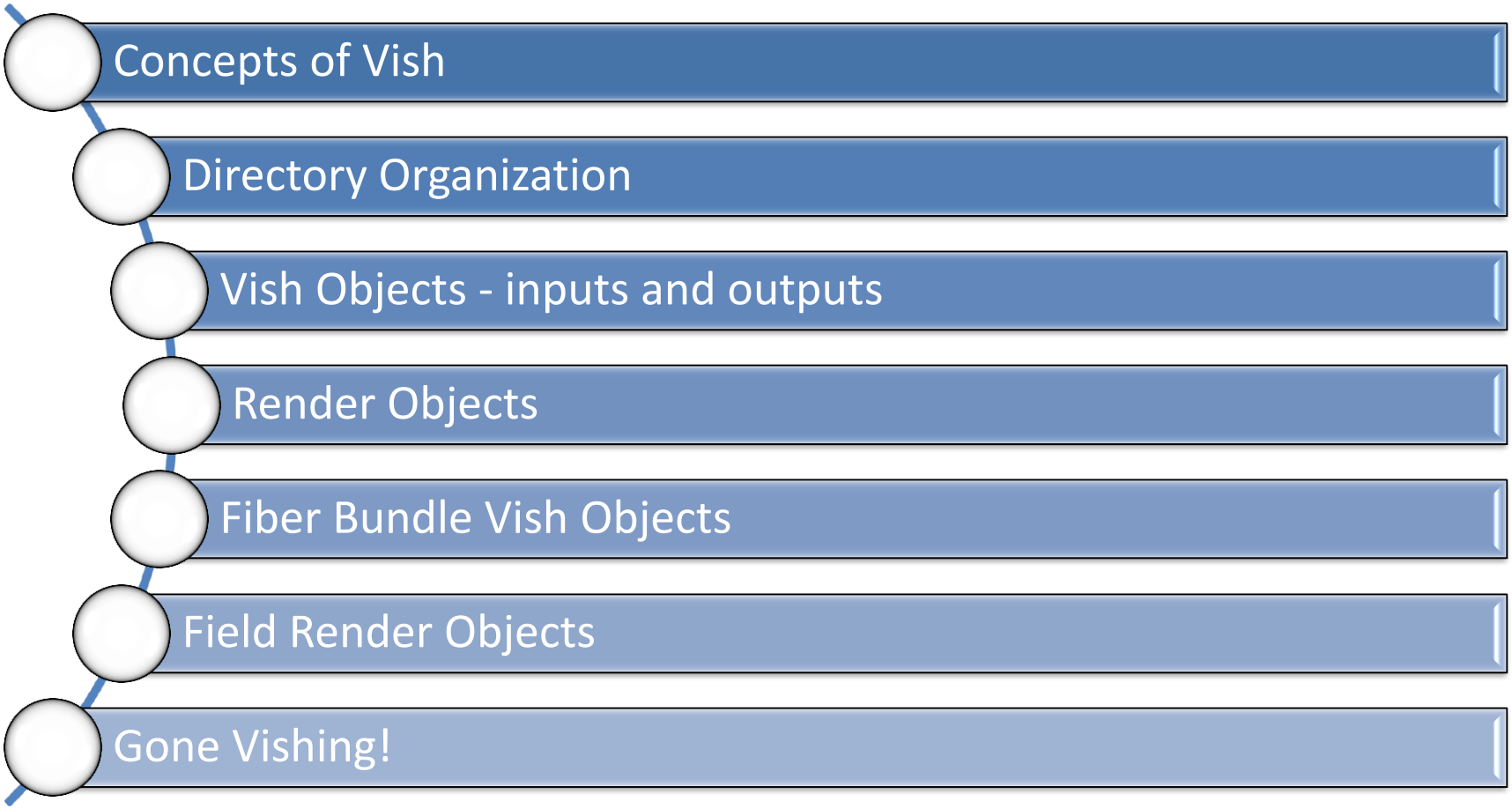
CSC 7700: Scientific Computing

Module D: Scientific Visualization

Lecture 4: Vish Coding

Dr. Werner Benger

Lecture Overview



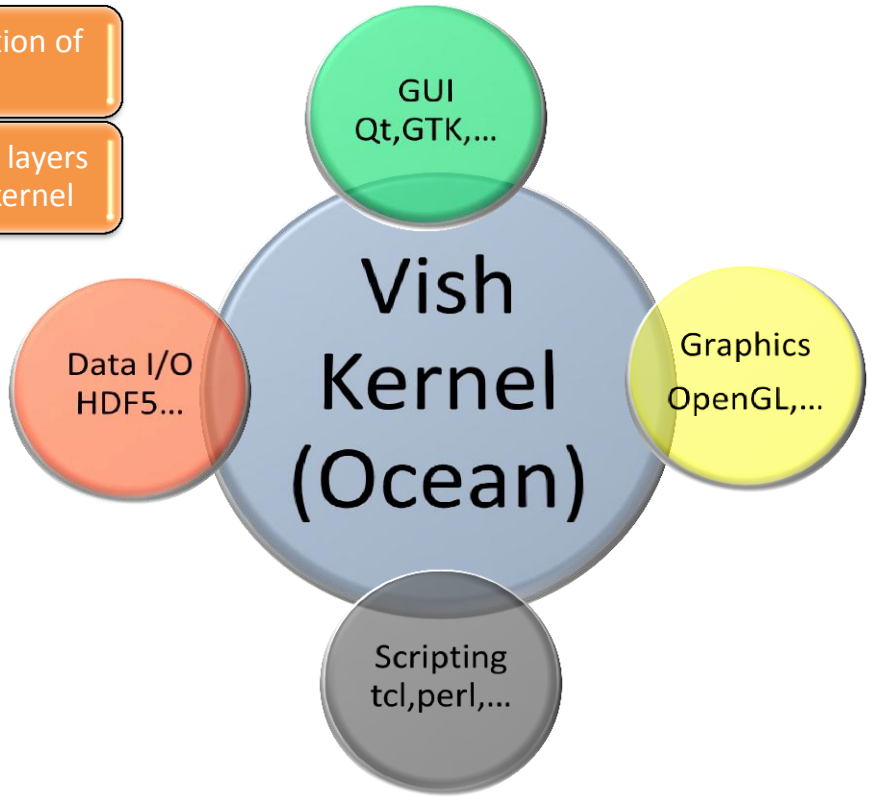
VISH ORGANIZATION

Visualization Pipeline

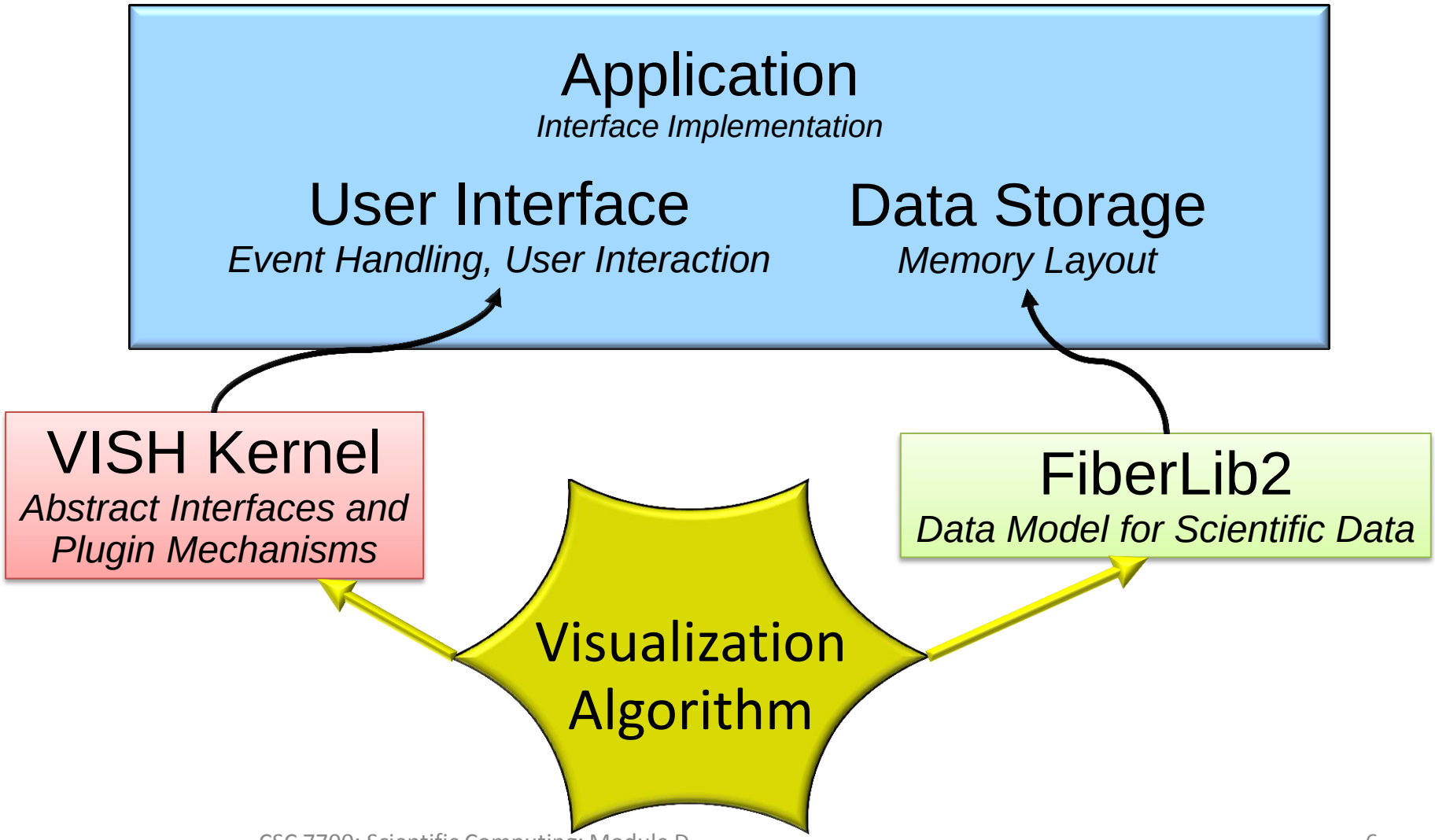
Implementing elements of the visualization pipeline.

Central Concept in Vish: *Encapsulation*

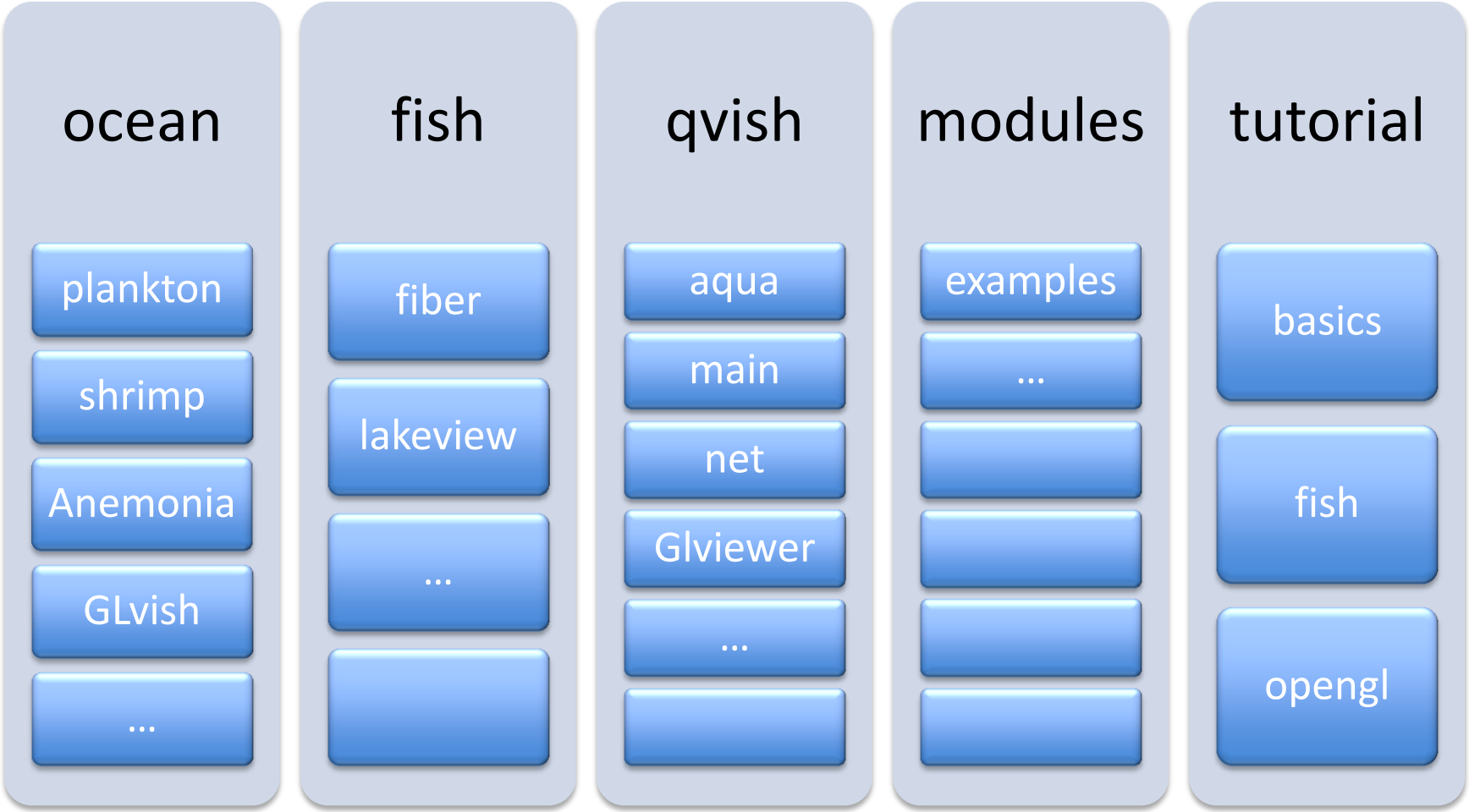
- No direct communication of components
- Hidden by abstraction layers provided by the Vish kernel



Vis Object only sees essential parts



Vish Directory Organization



Vish Ocean
Like a nutrition pyramid

VISH Kernel: ocean

Plankton - basic functionality implementing

- Abstract objects with inputs and outputs
- Data and control flow management
- Runtime plugin management

Shrimp – specific general-purpose objects

- Color, time, user interaction, bounding boxes, data ranges, 2D/3D positions

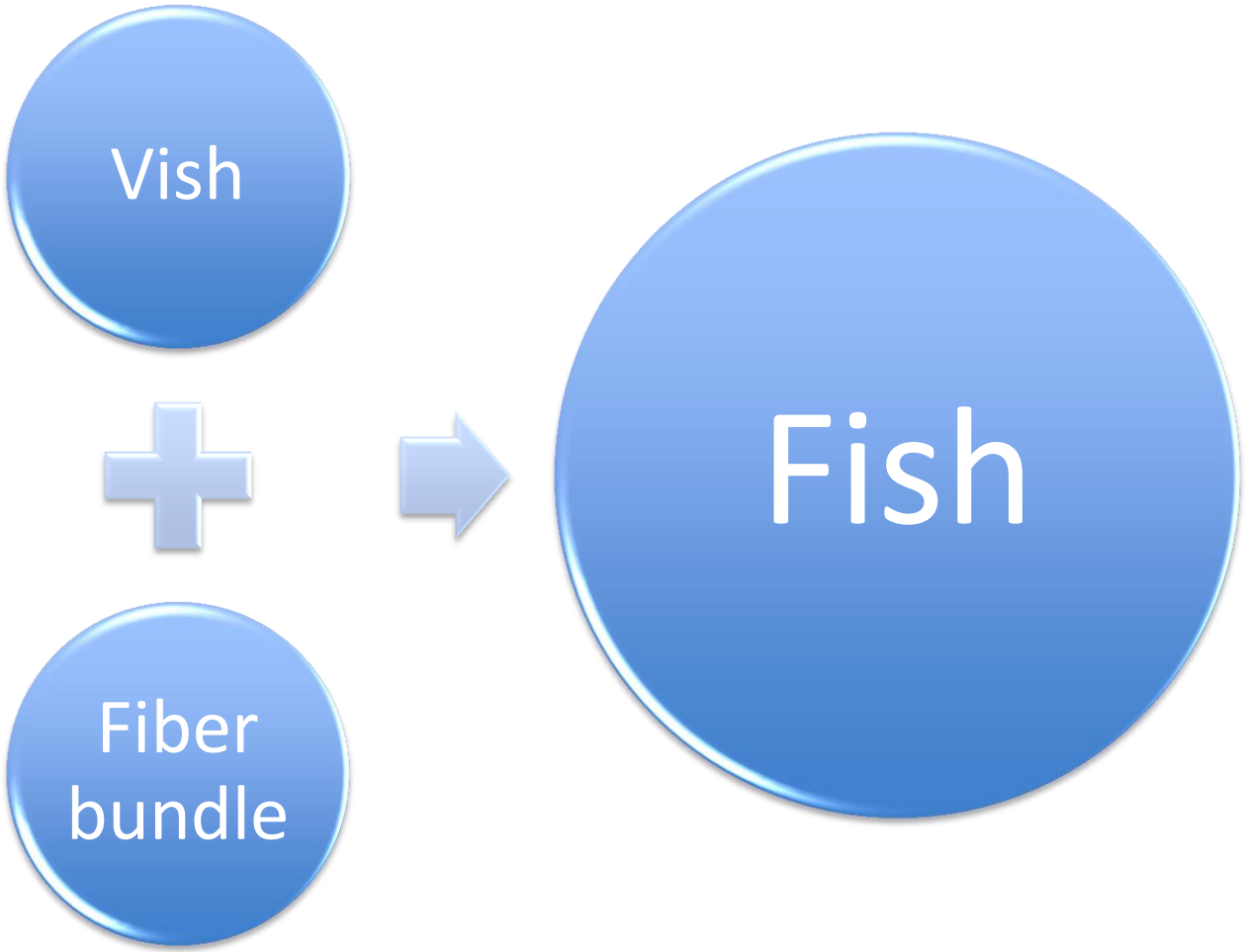
Anemonia – abstraction layer for rendering

- Graphics independent from OpenGL (work in progress)

GLvish - OpenGL support library

- C++ API for (subset) of OpenGL
- Management of OpenGL objects (OpenGL identifiers)
  - Allows caching and re-using OpenGL objects
  - Important for time-dependent data

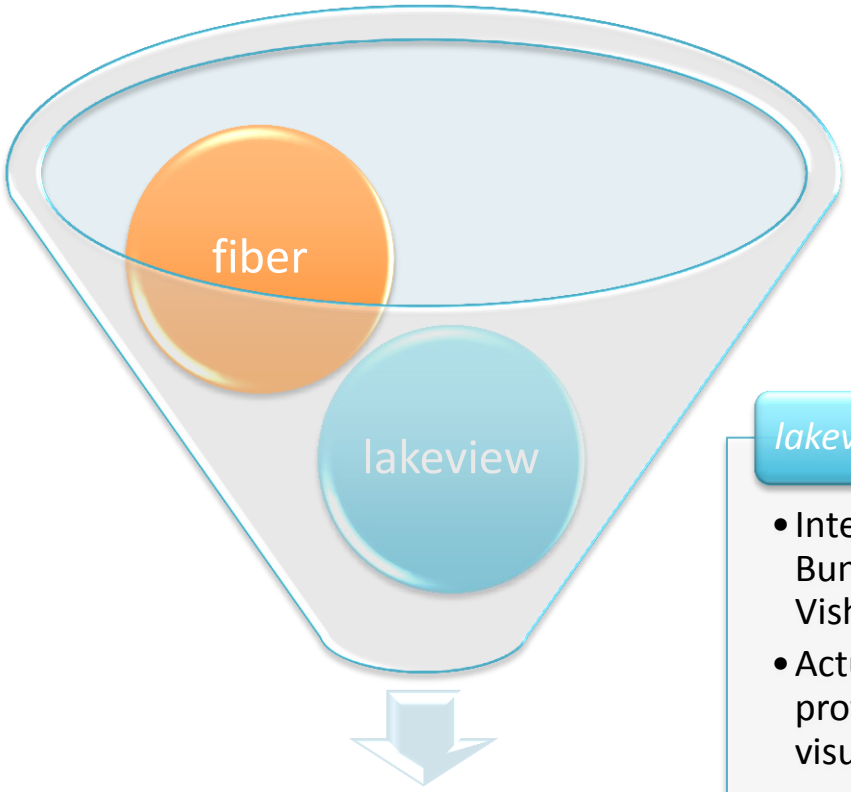
Data for Scientific Visualization



Structure of the Fish

Fiber

- Implementation of the Fiber Bundle data model
- Independent of Vish
- Collection of library functions that can be used outside of Vish
- I/O layers (reading/writing HDF5)



lakeview

- Interfacing the Fiber Bundle data model with Vish
- Actual "Fish" objects providing elements for the visualization pipeline

Vish Objects

Input and output parameters

Rendering

Context-relative evaluation

VISH CODING

Vish Objects

- Vish objects are derived from the base class “VObject”
- Need to define constructor (three arguments)
- Computational actions are invoked via a virtual update() function
- Can equip VObject with inputs and outputs
- Vish object will make use of basic or advanced libraries depending on functionality

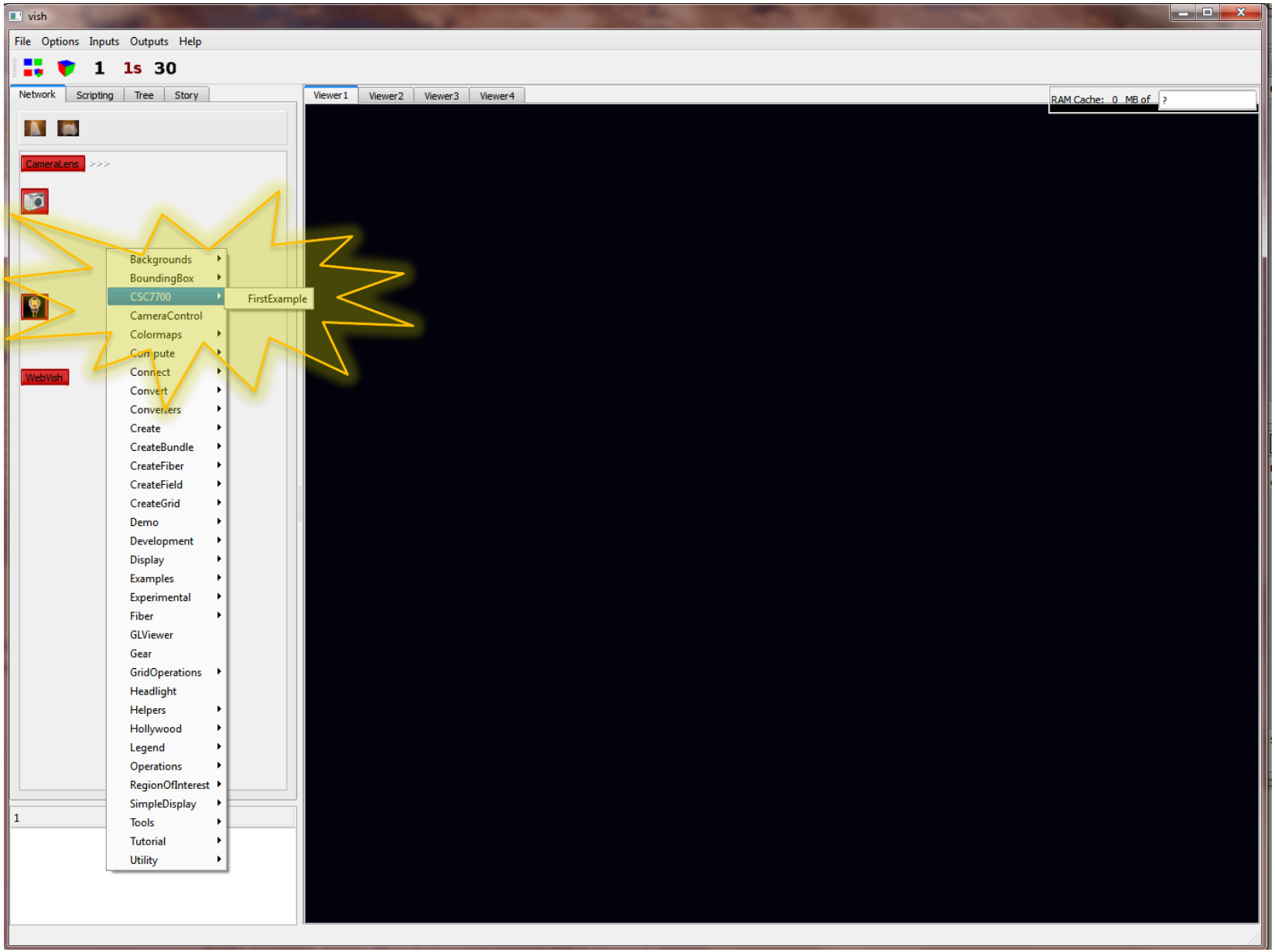
C++ Source: Basic Vish Example

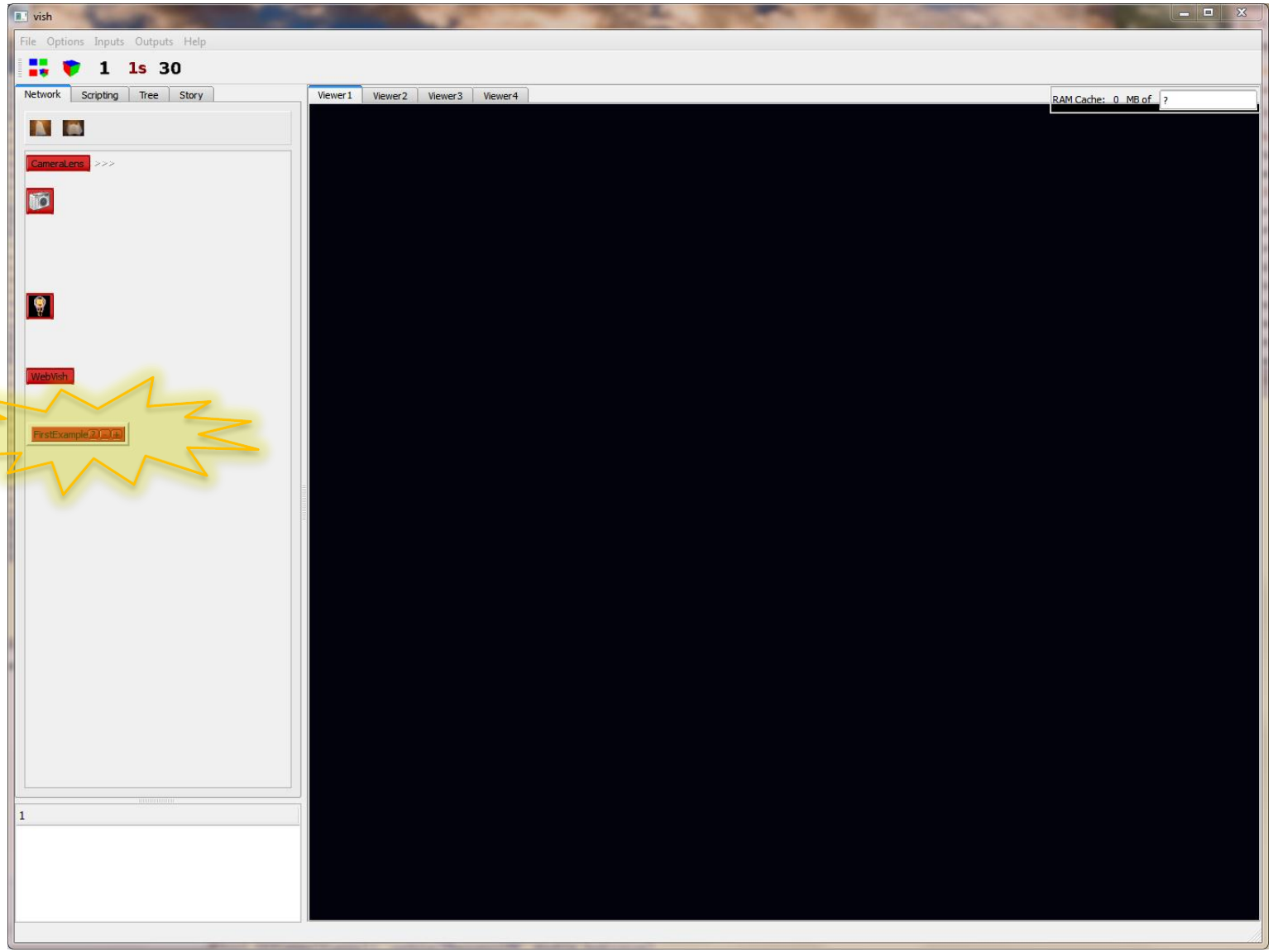
```
#include <ocean/plankton/VObject.hpp>
#include <ocean/plankton/VCreator.hpp>

using namespace Wizt;

namespace
{

class   ExampleObject : public VObject
{
public:
      ExampleObject(const string&name, int p, const RefPtr<VCreationPreferences>&VP)
      : VObject(name, p, VP)
      {}
};

static  Ref<VCreator<ExampleObject> > myBackground("CSC7700/FirstExample", ObjectQuality::EXPERIMENTAL);

}
```

Derive from VObject base class

VCreator: Announce object creation to Vish

How the new object should appear in the GUI and Script


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
CameraLens
WebVish
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
Backgrounds
BoundingBox
CSC7700
FirstExample
CameraControl
Colormaps
Compute
Connect
Convert
Converters
Create
CreateBundle
CreateFiber
CreateField
CreateGrid
Demo
Development
Display
Examples
Experimental
Fiber
GLViewer
Gear
GridOperations
Headlight
Helpers
Hollywood
Legend
Operations
RegionOfInterest
SimpleDisplay
Tools
Tutorial
Utility


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
CameraLens >>>
WebVish
FirstExample
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
1

Makefile

- Create subdir vish/modules/CSC7700/
- Create vish/modules/CSC7700/Makefile

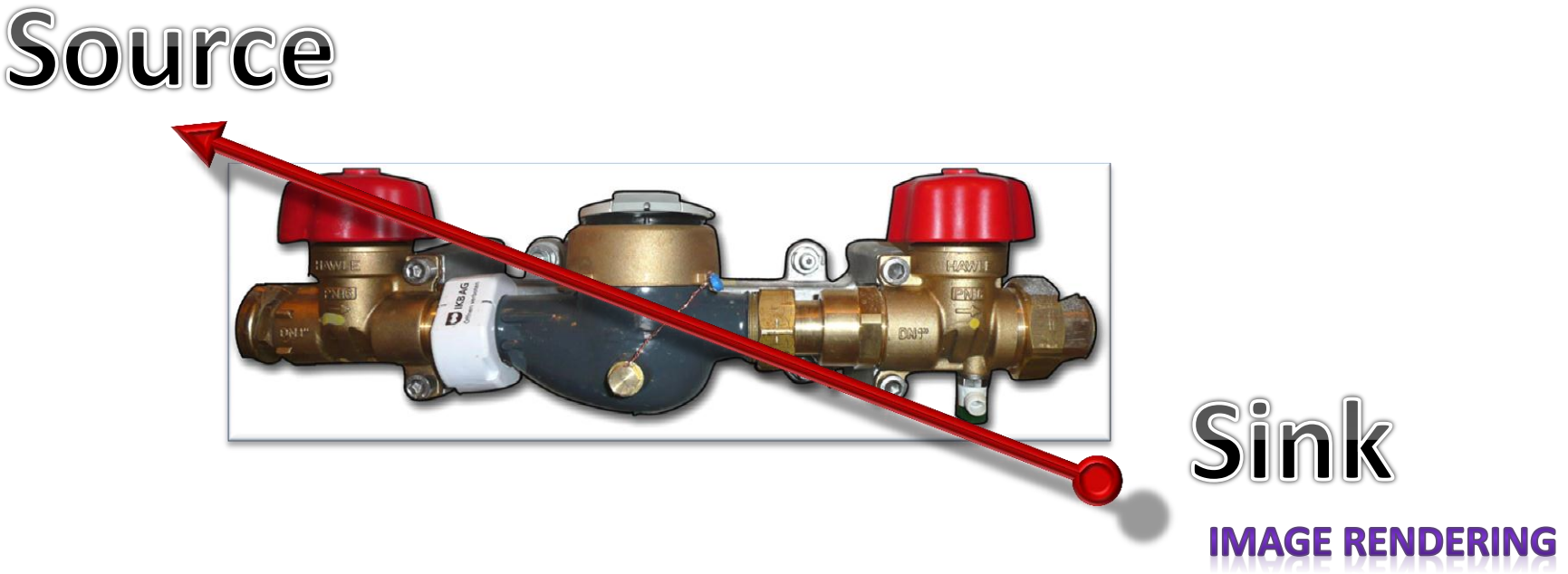
```
VISH = CSExample

LIBS=${OCEAN}

include $(VPATH)../GNUmakefile.rules
```

- Save C++ source code as vish/modules/CSC7700/SimpleVish.cpp
- Compile

```
$ cd vish/modules/CSC7700
vish/modules/CSC7700 $  make
```

Getting Started

- Vish implements the “pull” data flow model
- Need to have a *sink* to initiate any activity



OpenGL Render Objects - Data Sinks

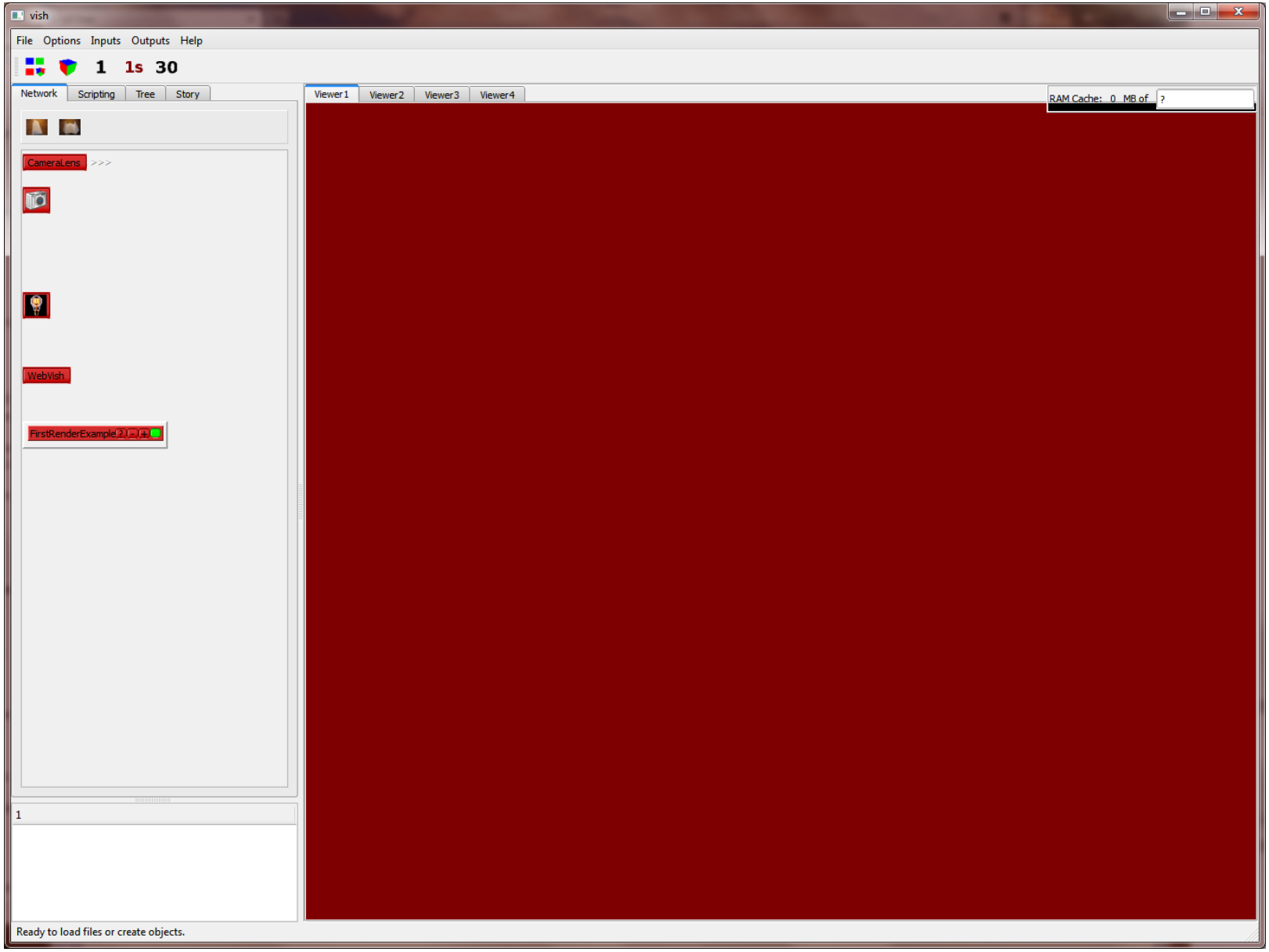
SimpleRender.cpp

```
#include <ocean/GLvish/VGLRenderObject.hpp>
#include <ocean/plankton/VCreator.hpp>

using namespace Wizt;

namespace
{

class    ExampleObject : public VGLRenderObject
{
public:
        ExampleObject(const string&name, int, const RefPtr<VCreationPreferences>&VP)
        : VGLRenderObject(name, DEFAULT_OBJECT, VP)
        {}

        override void render(VGLRenderContext&) const
        {
                glClearColor( 0.5, 0.0, 0.0, 1.0 );
                glClear(GL_COLOR_BUFFER_BIT);
        }
};

static    Ref<VCreator<ExampleObject> > myBackground("CSC7700/FirstRenderExample", ObjectQuality::EXPERIMENTAL);

}
```

Derive from VGLRenderObject base class

Implement render function
Calls of OpenGL functions are allowed here
See http://www.opengl.org/ for details.

http://www.opengl.org/sdk/docs/man/xhtml/glClearColor.xml
http://www.opengl.org/sdk/docs/man/xhtml/glClear.xml


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
CameraLens >>>
WebVish
FirstRenderExample
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
1
Ready to load files or create objects.

Vish Object Input Parameters

Input parameters are described using the

TypedSlot<T>

C++ template, where T is the *type* of the parameter, such as **int** or **double**, a C structure, or a user-defined type (details later).

Typed slots are evaluated relative to a given Context.

Render Objects with Parameter

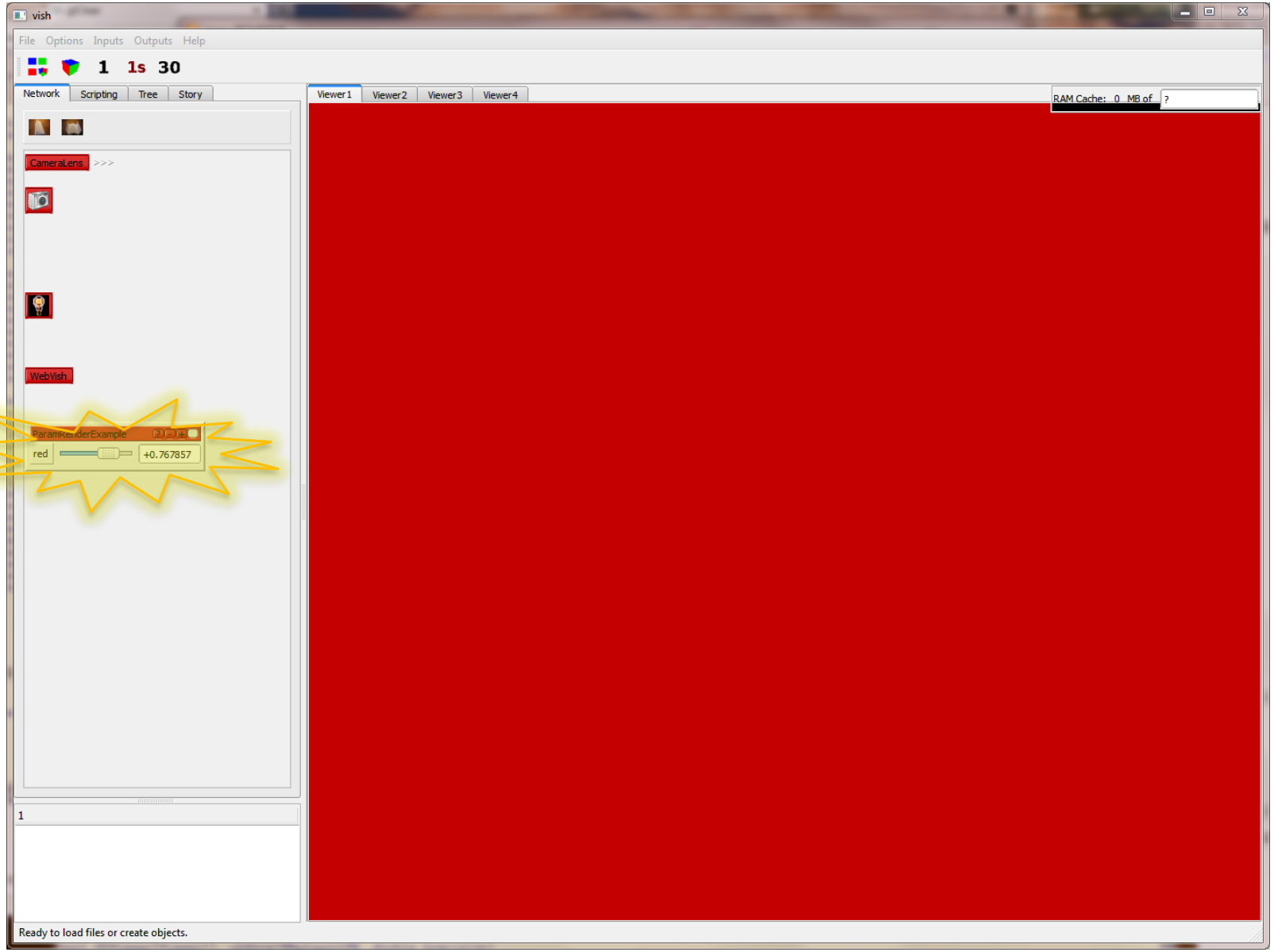
ParamRender.cpp

```
#include <ocean/GLvish/VGLRenderObject.hpp>
#include <ocean/plankton/VCreator.hpp>
using namespace Wizt;
namespace
{

class    ExampleObject : public VGLRenderObject
{
        TypedSlot<double>    RedIntensity;
public:
        ExampleObject(const string&name, int, const RefPtr<VCreationPreferences>&VP)
        : VGLRenderObject(name, DEFAULT_OBJECT, VP)
        , RedIntensity(this, "red", 0.5)
        {}

        override void render(VGLRenderContext&Context) const
        {
        double  red = 0.5;
                RedIntensity << Context >> red;
                glClearColor( 0.5, 0.0, 0.0, 1.0 );
                glClear(GL_COLOR_BUFFER_BIT);
        }
};

static   Ref<VCreator<ExampleObject> > myBackground("CSC7700/ParamRenderExample", ObjectQuality::EXPERIMENTAL);
}
```

Input parameter of type “double”

Define name of parameter and initial value

Evaluate relative to a given Context


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
CameraLens >>>
WebVish
ParamRenderExample
red
+0.767857
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
1
Ready to load files or create objects.

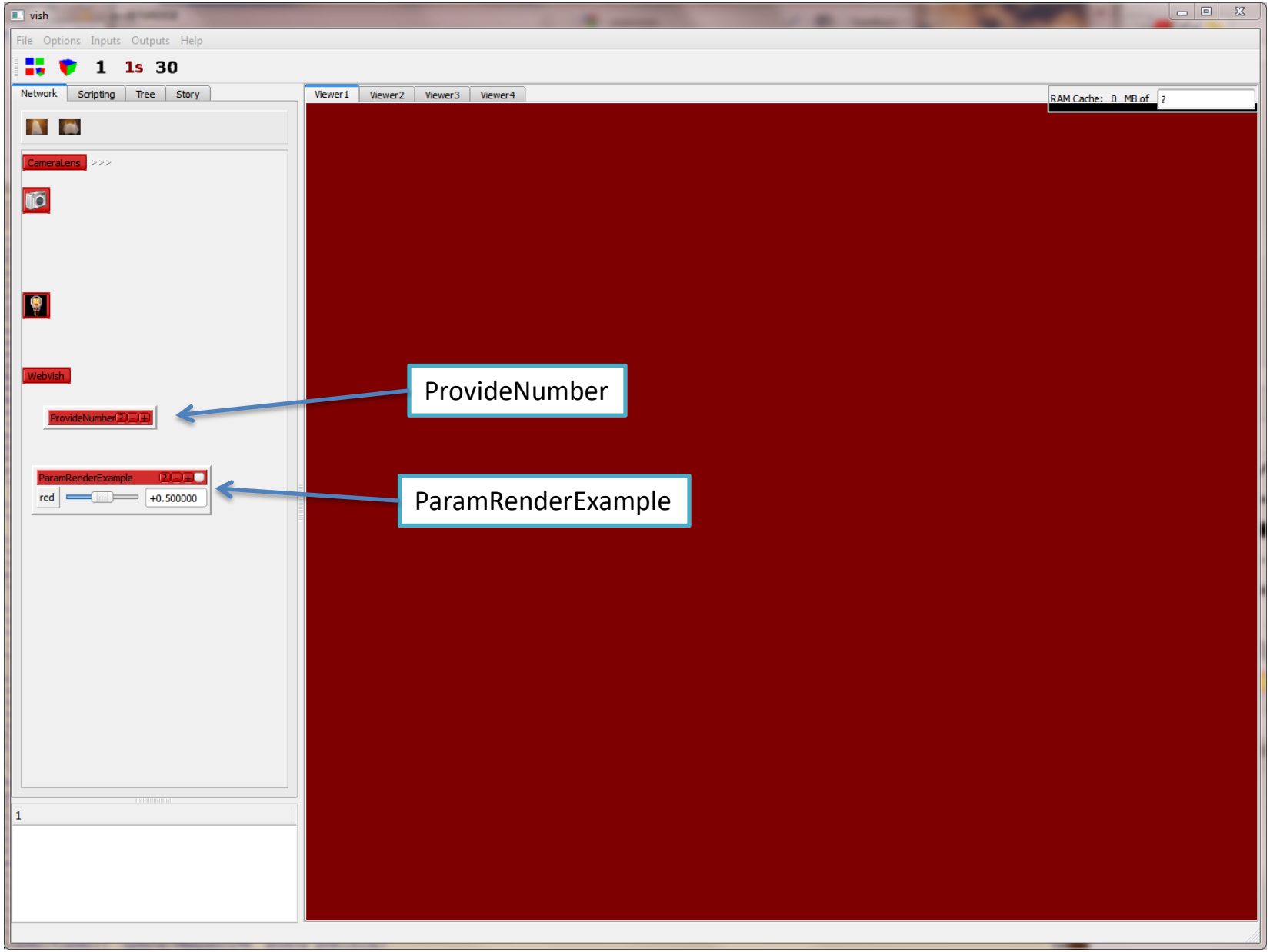
Vish Objects providing Outputs

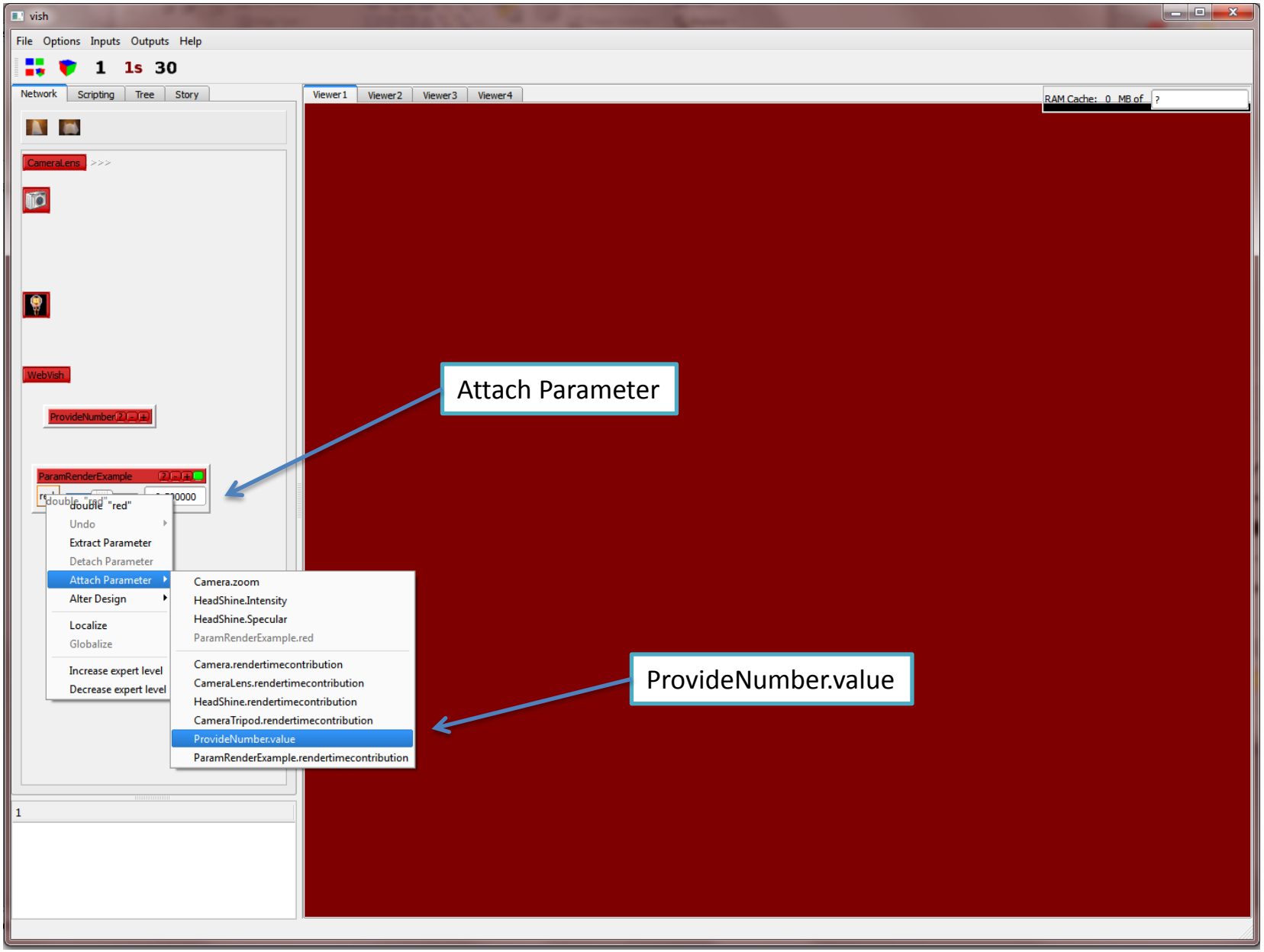
ProvideValue.cpp

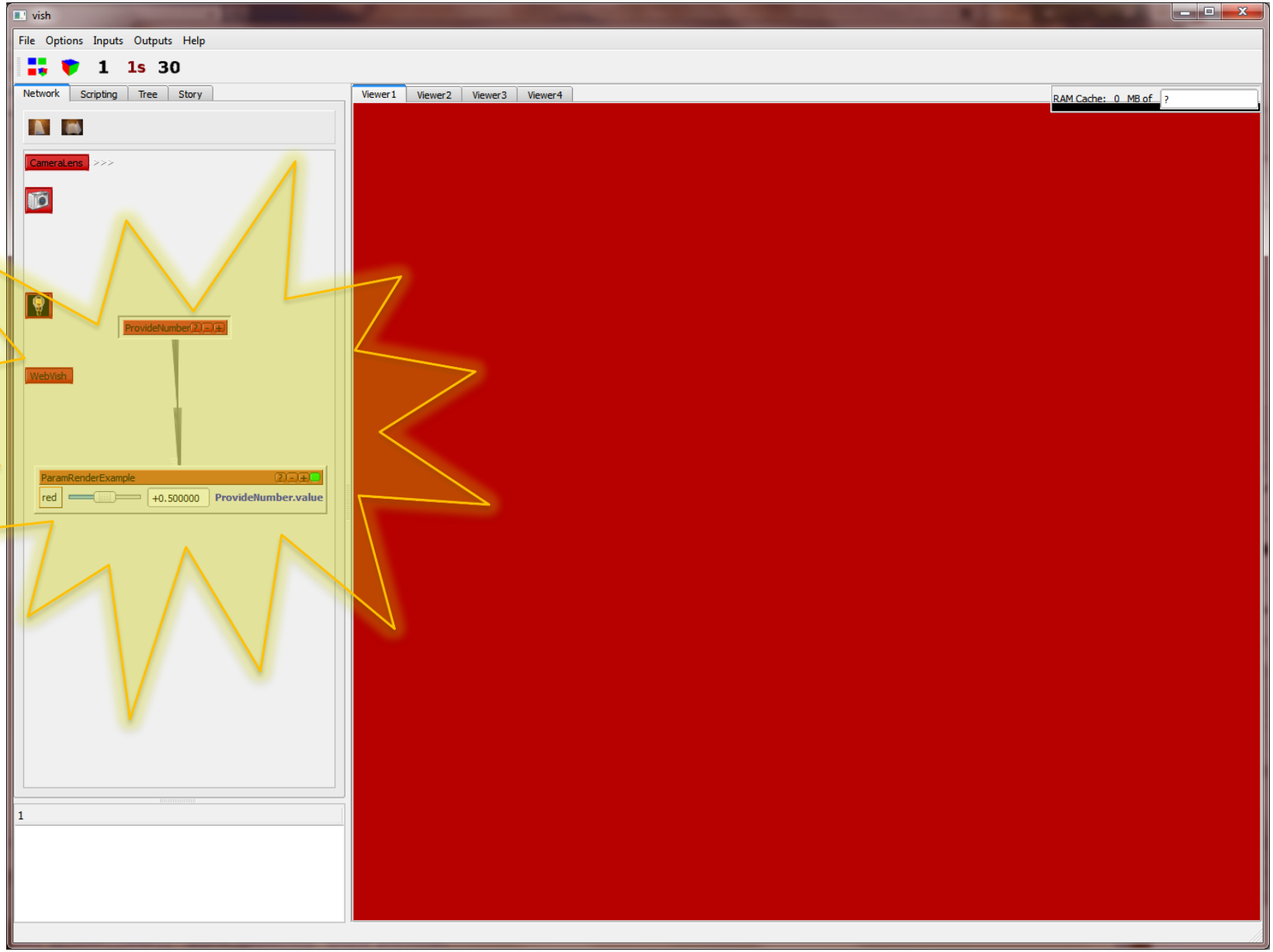
```
#include <ocean/plankton/VObject.hpp>
#include <ocean/plankton/VCreator.hpp>

using namespace Wizt;

namespace
{

class    ExampleObject : public VObject
{
public:
        VOutput<double>     OutputValue;

        ExampleObject(const string&name, int p, const RefPtr<VCreationPreferences>&VP)
        : VObject(name, p, VP)
        , OutputValue( self(), "value", 0.717 )
        {}
};

static   Ref<VCreator<ExampleObject> > myBackground("CSC7700/ProvideNumber", ObjectQuality::EXPERIMENTAL);

}
```

Output parameter of type “double”

Initialize output parameter of type “double”

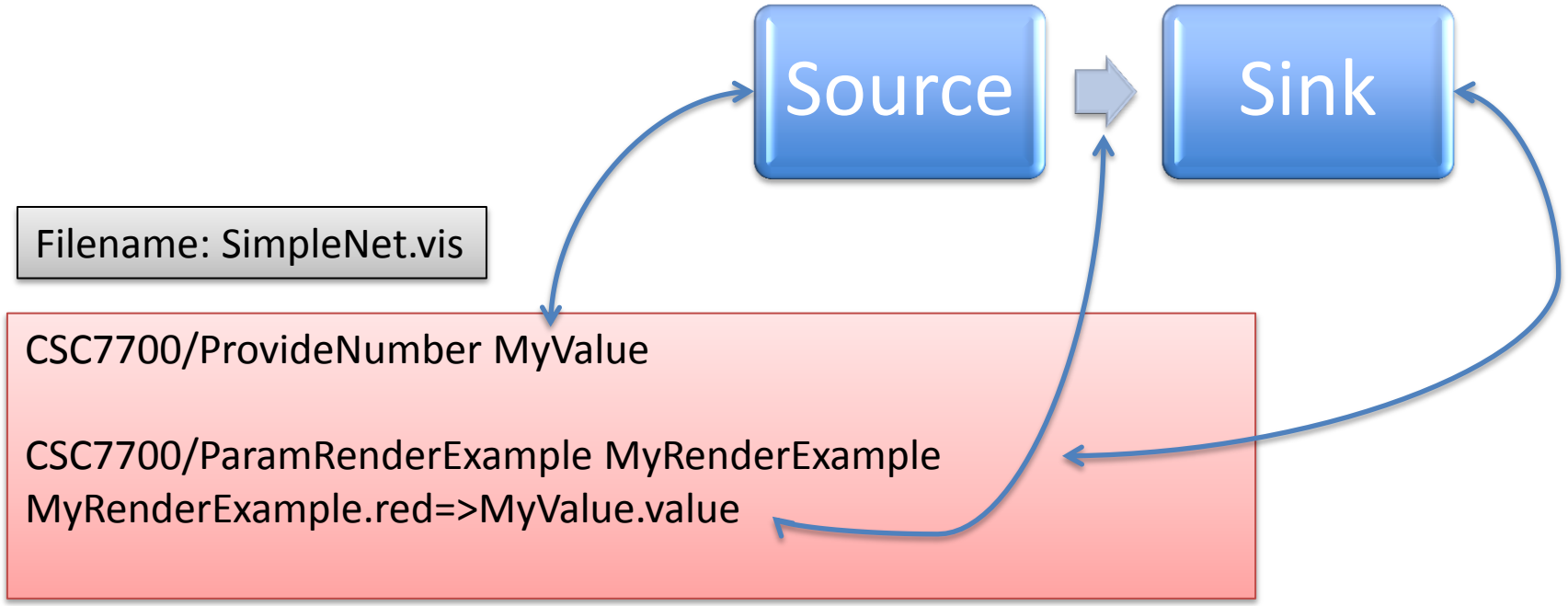

vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
CameraLens >>>
WebVish
ProvideNumber
ParamRenderExample
red
+0.500000
1
ProvideNumber
ParamRenderExample


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
CameraLens
WebVish
ProvideNumber
ParamRenderExample
double "red"
Undo
Extract Parameter
Detach Parameter
Attach Parameter
Alter Design
Localize
Globalize
Increase expert level
Decrease expert level
Camera.zoom
HeadShine.Intensity
HeadShine.Specular
ParamRenderExample.red
Camera.rendertimecontribution
CameraLens.rendertimecontribution
HeadShine.rendertimecontribution
CameraTripod.rendertimecontribution
ProvideNumber.value
ParamRenderExample.rendertimecontribution
Attach Parameter
ProvideNumber.value


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
CameraLens >>>
ProvideNumber2
WebVish
ParamRenderExample
red
+0.500000
ProvideNumber.value
1
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?

Just implemented a Viz Network

- Use script to reproduce



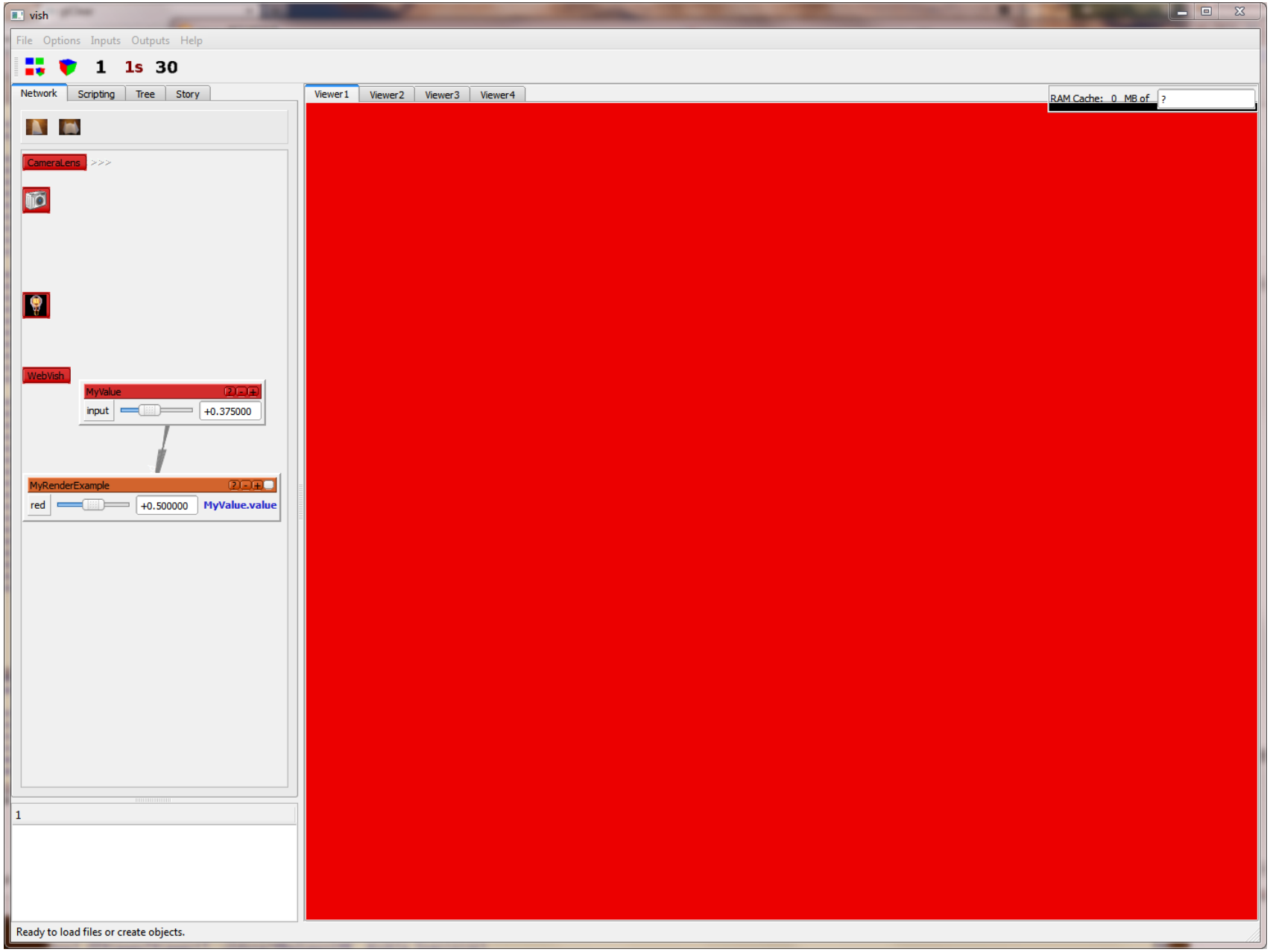
Objects performing Operations

SinusValue.cpp

```
#include <ocean/plankton/VObject.hpp>
#include <ocean/plankton/VCreator.hpp>
#include <math.h>
using namespace Wizt;

namespace
{

class   ExampleObject : public VObject
{
public:
        TypedSlot<double>       InputValue;
        VOutput<double>         OutputValue;

        ExampleObject(const string&name, int p, const RefPtr<VCreationPreferences>&VP)
        : VObject(name, p, VP)
        , InputValue( this, "input", 0.0)
        , OutputValue( self(), "value", 0.0 )
        {}

        bool update(VRequest&Context, double precision)
        {
        double  in = 0.0;
                InputValue << Context >> in;

        double  result = sin( 3.141692*in);
                OutputValue << Context << result;

                return true;
        }
};
static  Ref<VCreator<ExampleObject> > myBackground("CSC7700/SinusValue", ObjectQuality::EXPERIMENTAL);
}
```

update() function
- evaluates input parameters
- performs operations
- sets out parameters

(relative to a *Context*)


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
CameraLens >>>
WebVish
MyValue
input +0.375000
MyRenderExample
red +0.500000 MyValue.value
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
1
Ready to load files or create objects.

Operations in Render Objects

SinusRender.cpp

```
#include <ocean/GLvish/VGLRenderObject.hpp>
#include <ocean/plankton/VCreator.hpp>
#include <math.h>
using namespace Wizt;
namespace
{
class	ExampleObject : public VGLRenderObject
{
	TypedSlot<double>		RedIntensity;
	double				RenderValue;
public:
	ExampleObject(const string&name, int, const RefPtr<VCreationPreferences>&VP)
	: VGLRenderObject(name, DEFAULT_OBJECT, VP)
	, RedIntensity(this, "red", 0.5)
	, RenderValue(0.0)
	{}

	override bool update(VRequest&Context, double precision)
	{
	double	red = 0.5;
		RedIntensity << Context >> red;

		RenderValue = sin( red*3.141592 );
		return true;
	}

	override void render(VGLRenderContext&Context) const
	{
		glClearColor( RenderValue, 0.0, 0.0, 1.0 );
		glClear(GL_COLOR_BUFFER_BIT);
	}
};
static	Ref<VCreator<ExampleObject> > myBackground("CSC7700/SinusRender", ObjectQuality::EXPERIMENTAL);
}
```

update()
supposed to do *computationally intensive* tasks

render()
supposed to do *only rendering*

Computing & Rendering

update()

- is called only when object parameters change
- Few calls
- May perform computationally intensive (slow) operations
- May modify the Vish object

render()

- is called anytime when the mouse is moved, window is resized, another object is rendered…
- Frequent calls
- Should only do fast rendering operations
- Must not modify the Vish object

SinusStateRender.cpp

Context-Relative Values

Problem: communication of values between update() and render() ought to be context-relative:

- Computational results of update() should be stored in a "State" object that exists for each "Context"
- State objects are created by the Vish kernel on request
- User class needs to define the concrete layout of data members

```
class   ExampleObject : public VGLRenderObject
{
        TypedSlot<double> RedIntensity;

        struct    MyState : State
        {
                double    RenderValue;

                MyState()
                : RenderValue(0.0)
                {}
        };

        override RefPtr<State> newState() const
        {
                return new MyState();
        }

public:
        ExampleObject(const string&name, int, const RefPtr<VCreationPreferences>&VP)
        : VGLRenderObject(name, DEFAULT_OBJECT, VP)
        , RedIntensity(this, "red", 0.5)
        {}

        override bool update(VRequest&Context, double precision)
        {
        double    red = 0.5;
                RedIntensity << Context >> red;

        RefPtr<MyState> theState = myState(Context);
                theState->RenderValue = sin( red*3.141592 );
                return true;
        }

        override void render(VGLRenderContext&Context) const
        {
        RefPtr<MyState> theState = myState(Context);
                glClearColor( theState->RenderValue, 0.0, 0.0, 1.0 );
                glClear(GL_COLOR_BUFFER_BIT);
        }
};
```

Define object-specific state object

Write to state object

Read from state object

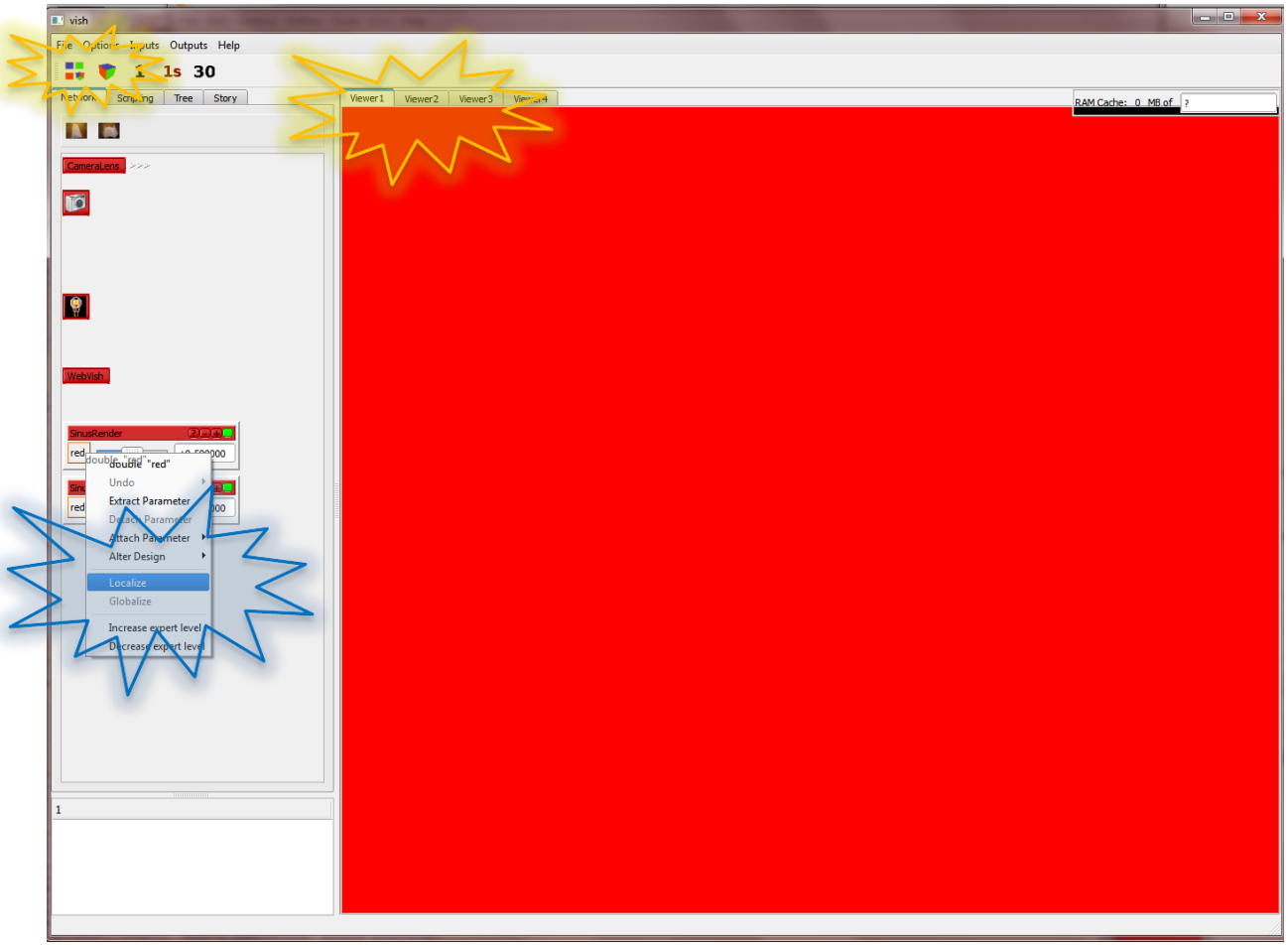
What *is* a context?

Values can be set relative to a Viewer

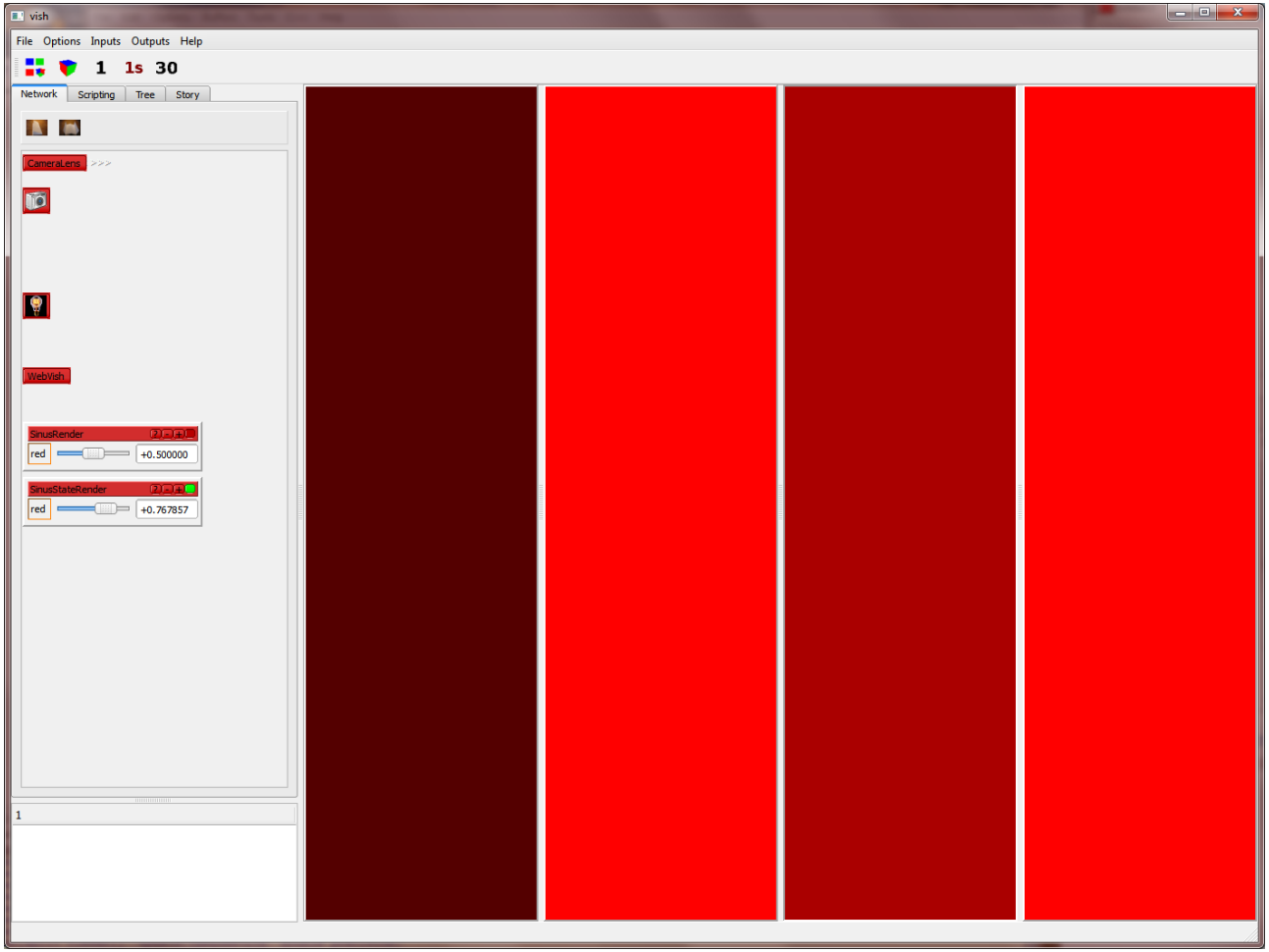
Localize vs. globalize

Full viewer vs. splitted viewer

Allows to explore parameter spaces



Same object – different parameters



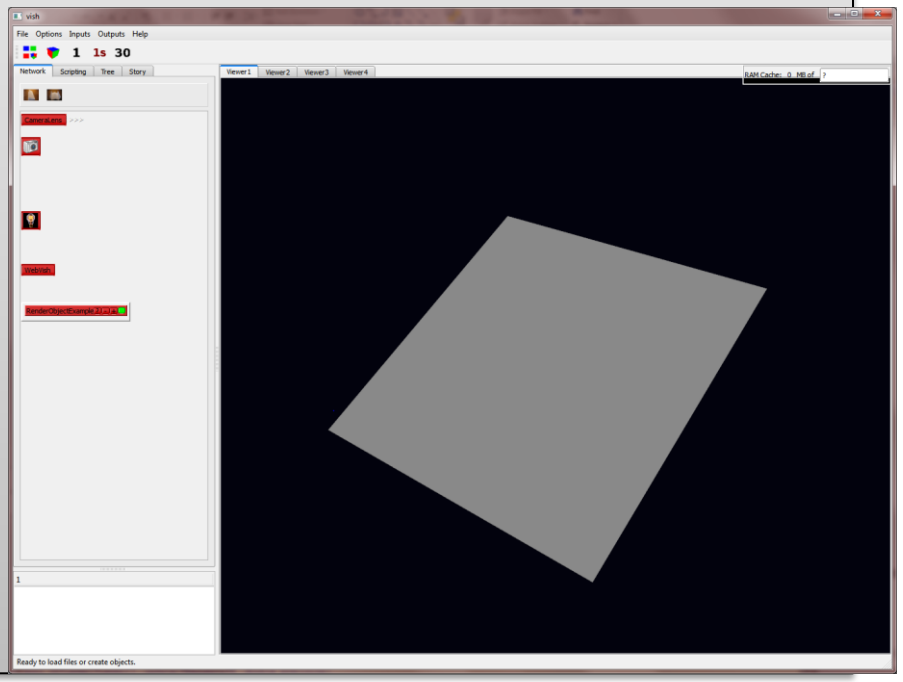
Rendering 3D Geometries

Render3DObject.cpp

```
class   ExampleObject : public VGLRenderObject
{
public:
        ExampleObject(const string&name, int, const RefPtr<VCreationPreferences>&VP)
        : VGLRenderObject(name, DEFAULT_OBJECT, VP)
        {}

        override bool update(VRequest&Context, double precision)
        {
        Eagle::PhysicalSpace::point Min, Max;
                Min =  0,  0,  0;
                Max =  1,  1,  1;

                setBoundingBall( Context, new BoundingBox(Min, Max) );
                return true;
        }

        override void render(VGLRenderContext&Context) const
        {
                glBegin(GL_QUADS);
                glVertex3f(0,0,0);
                glVertex3f(1,0,0);
                glVertex3f(1,1,0);
                glVertex3f(0,1,0);
                glEnd();
        }
};
```

Need to set bounding box information for geometry

OpenGL code *only* in render()!



Modern OpenGL Vertex Arrays

```
override void render(VGLRenderContext&Context) const
{
Eagle::PhysicalSpace::point Vertices[4];
        Vertices[0] = 0,0,0;
        Vertices[1] = 1,0,0;
        Vertices[2] = 1,1,0;
        Vertices[3] = 0,1,0;

        // http://www.opengl.org/sdk/docs/man/xhtml/glVertexPointer.xml
        glVertexPointer( 3, GL_DOUBLE, 0, &Vertices);

        glEnableClientState(GL_VERTEX_ARRAY);

GLushort VertexIndices[] = {0,1,2,3};

        // http://www.opengl.org/sdk/docs/man/xhtml/glDrawElements.xml
        glDrawElements( GL_QUADS, 4, GL_UNSIGNED_SHORT, VertexIndices );

        glDisableClientState(GL_VERTEX_ARRAY);
}
```

Modern OpenGL prefers array operations instead of procedural descriptions glBegin()/glEnd()

This basically means support for fiber bundles in modern OpenGL and on GPU's!

Old procedural OpenGL is considered "deprecated" and might no longer be supported, though in practice it will be there for a long time.

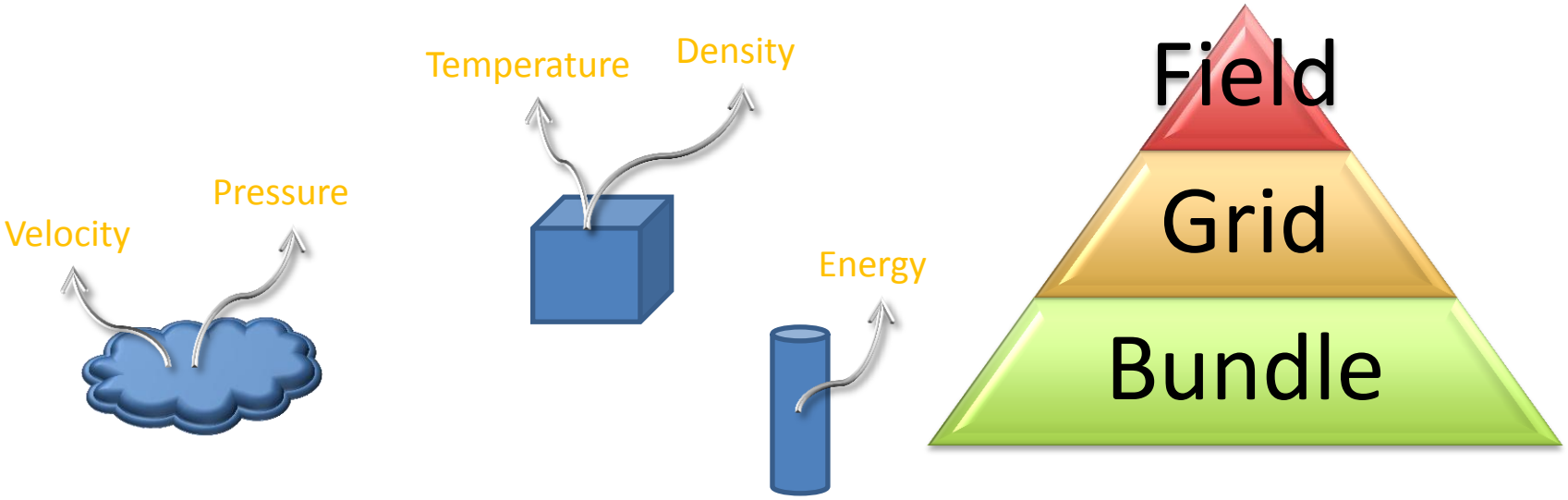
OpenGL-ES no longer supports procedural calls, only array (fiber-bundle) operations (used in WebGL and embedded systems such as cell phones).

Operating on fiber-bundle data

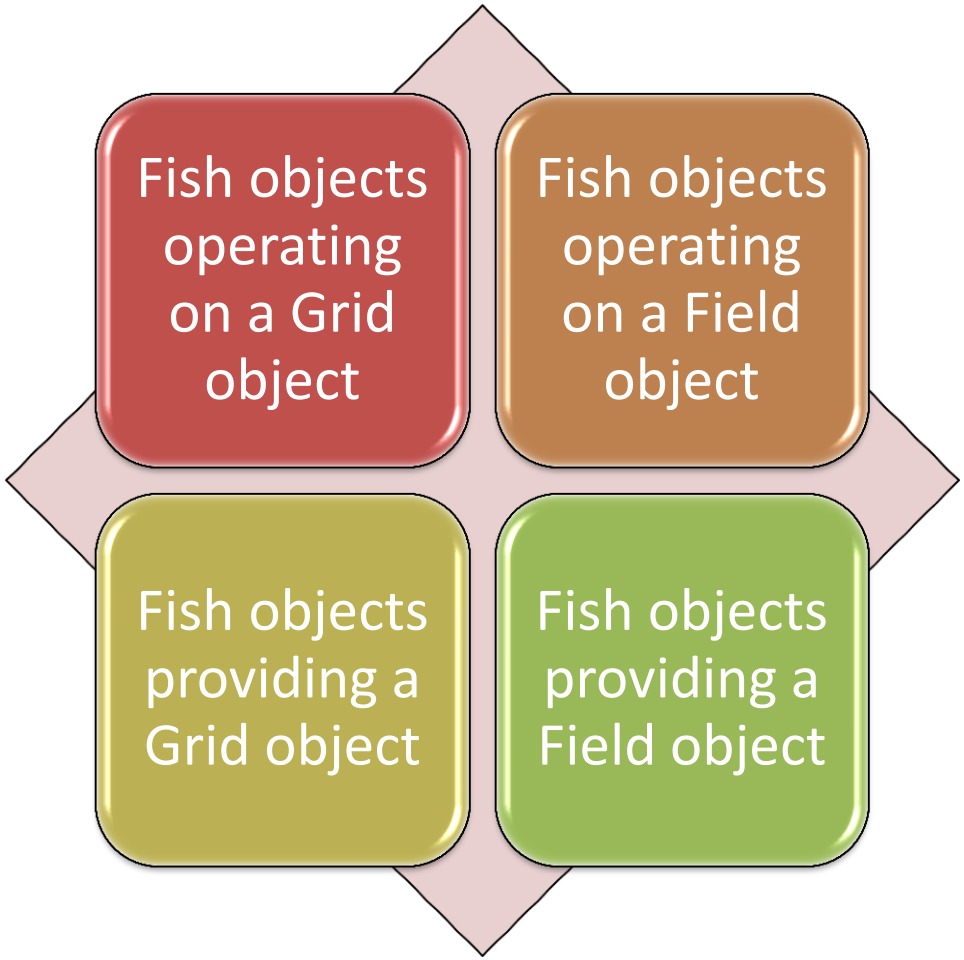
FIBER-VISH

Fish Objects

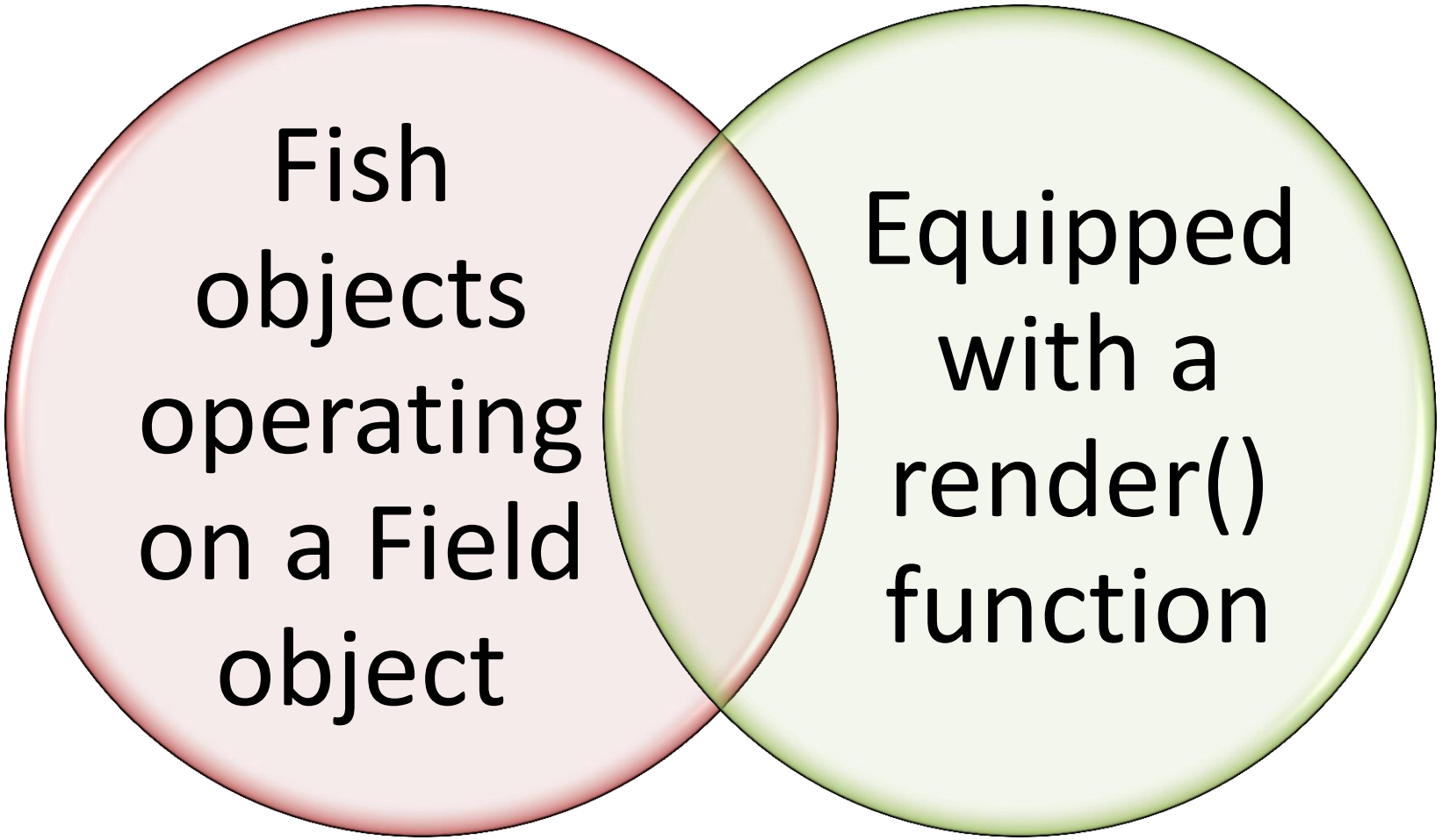
- Vish Objects operating on data stored in the fiber bundle and communicate fiber-bundle data
- There are exactly three choices of "communication types" available: *Bundle*, *Grid* and *Field* objects



Types of Fish Objects

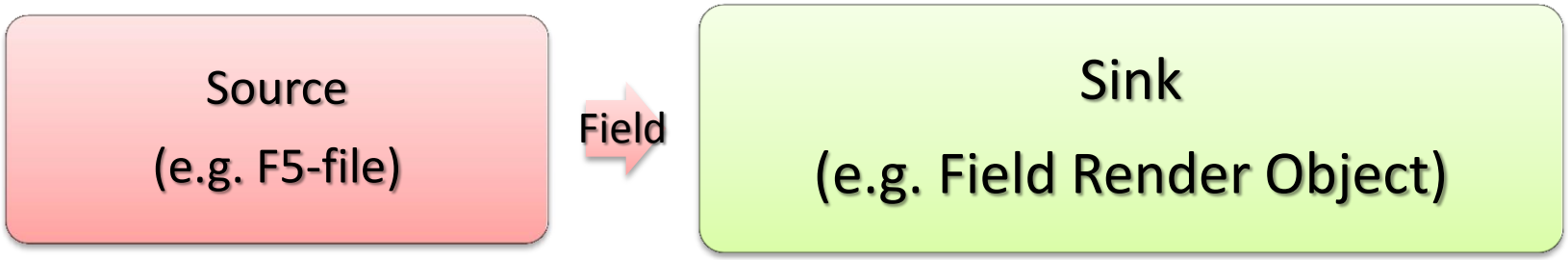


Render a Field



Operating on a Field

- Fields are basically used via a **TypedSlot<Field>**
  - (actual implementation communicates "FieldSelector" instead of Field objects for performance reasons)
- Convenience class **Fish<Field>** provides additional parameters, such as time selection (select one Field instance out of a time series)



Basic Fish<Field> Object

CSC7700Fiber/UseAField.cpp

```
#include <ocean/plankton/VObject.hpp>
#include <ocean/plankton/VCreator.hpp>
#include <ocean/plankton/ObjectQuality.hpp>

#include <fish/lakeview/bone/FishField.hpp>
#include <fish/lakeview/bone/FishSlice.hpp>

namespace
{
using namespace Wizt;
using namespace Fiber;

class	MyObject :	public virtual VObject,
			public virtual Fish<Field>
{
public:
	MyObject (const string&name, int Quality, const RefPtr<VCreationPreferences>&vp)
	: VObject(name, Quality,vp )
	, Fish<VObject>( this )
	, Fish<Field>("inputfield")
	{}

	override bool update(VRequest&Context, double precision)
	{
		if (RefPtr<Field>  F = getField(Context) )
		{
			// process the Field F
		}
		return true;
	}
};

static Ref<VCreator<MyObject, AcceptList<Fiber::Field> > >
	SimpleCreatorThatUsesAnyField("CSC7700/UseAField", ObjectQuality::EXPERIMENTAL );

}
```

Field

Sink

(Field Object)

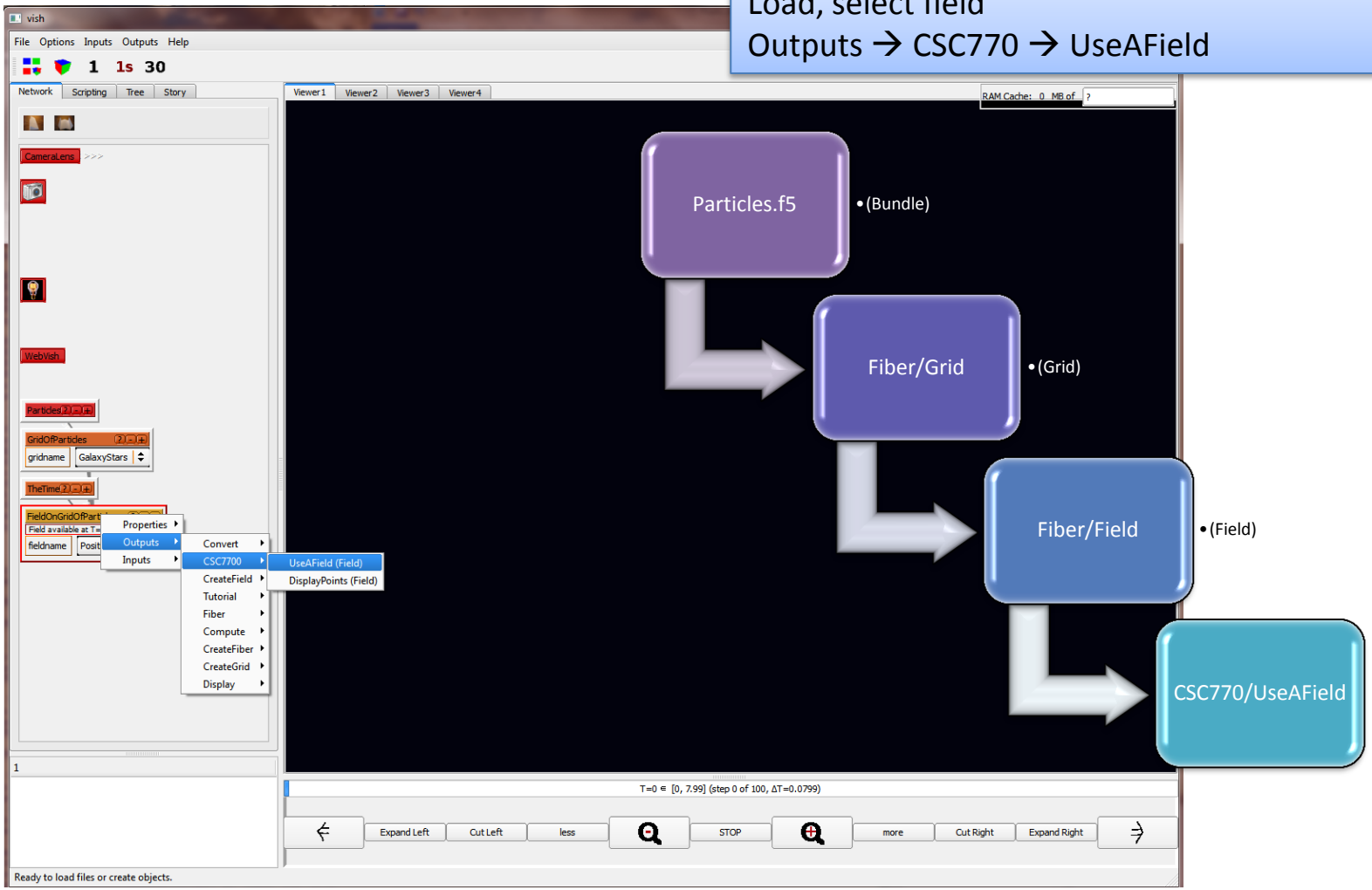
Initialize base class (takes care of TypedSlot<Field> input parameter)

Get actual Field object

Announce to Vish that this object likes Field's as inputs

Testing

Use data file data/Particles.f5
Load, select field
Outputs → CSC770 → UseAField



Fish<Field> Render Object

CSC7700Fiber/RenderField.cpp

```
#include <ocean/plankton/VObject.hpp>
#include <ocean/plankton/VCreator.hpp>
#include <ocean/GLvish/VGLRenderObject.hpp>

#include <fish/lakeview/bone/FishField.hpp>
#include <fish/lakeview/bone/FishSlice.hpp>

namespace
{
using namespace Wizt;
using namespace Fiber;

class	MyObject : public virtual VGLRenderObject,
			public virtual Fish<Field>
{
public:
MyObject (const string&name, int Quality, const RefPtr<VCreationPreferences>&vp)
	: VGLRenderObject(name, DEFAULT_OBJECT,vp )
	, Fish<VObject>( this )
	, Fish<Field>("inputfield")
	{}

	override void render(VGLRenderContext&Context) const
	{
		if (RefPtr<Field>  F = getField(Context) )
		{
			// process the Field F, do OpenGL here
		}
	}
};

static Ref<VCreator<MyObject, AcceptList<Fiber::Field> > >
	SimpleCreatorThatUsesAnyField("CSC7700/RenderField", ObjectQuality::EXPERIMENTAL );

}
```

Rendering a Field – the "quick way"

Derive from VGLRenderObject

render() function

Access Data & Render

CSC7700Fiber/DisplayPoints.cpp

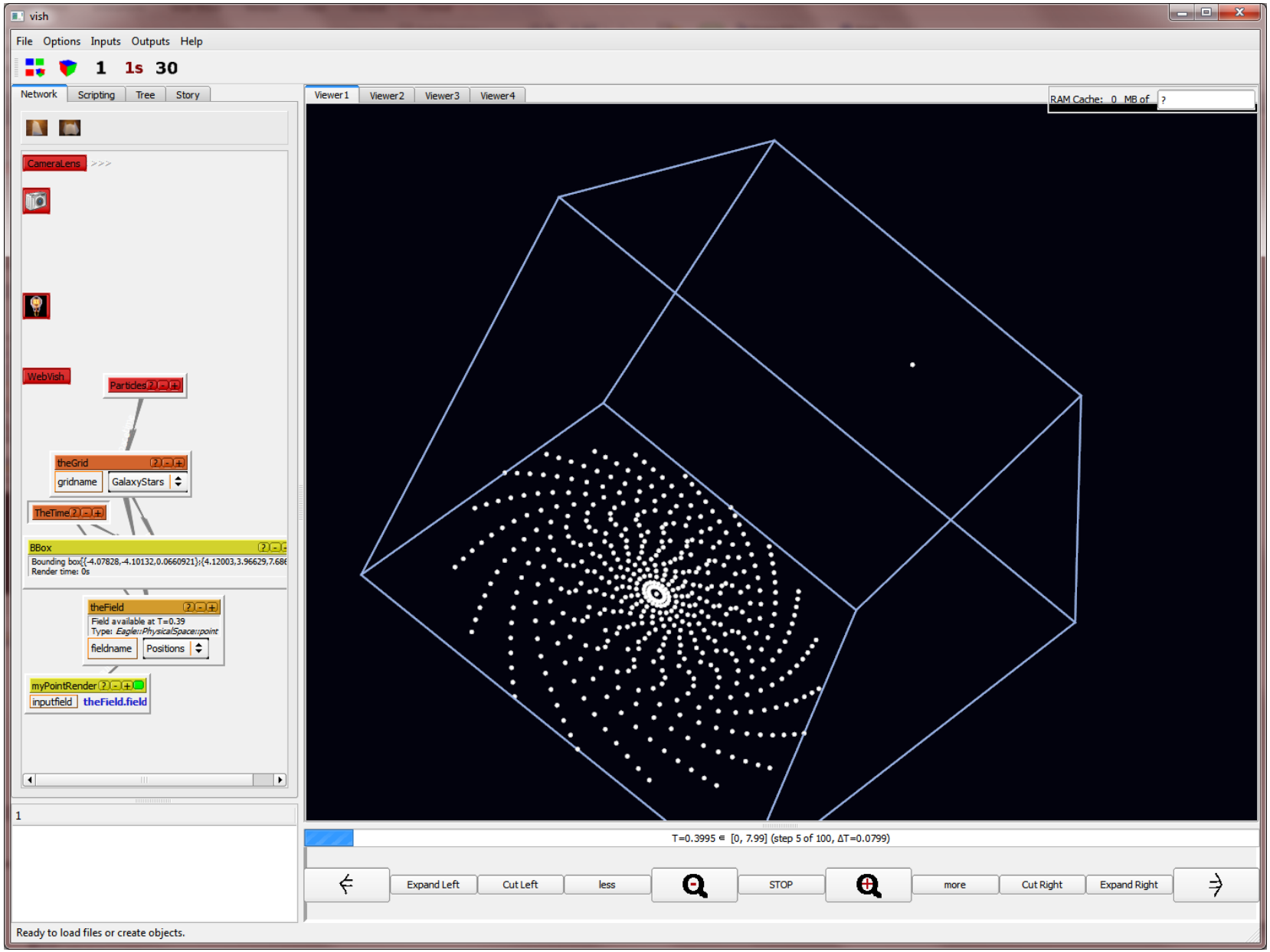
```
override void render(VGLRenderContext&Context) const
{
        if (RefPtr<Field>  F = getField(Context) )
        {
        using namespace Eagle;
                if (RefPtr<MemArray<1, PhysicalSpace::point> >
                   Vertices = F->getData() )
                {
                PhysicalSpace::point*pts = Vertices->ptr();
                unsigned NumberOfVertices = Vertices-> nElements();

                        glPointParameterf(GL_POINT_SIZE_MAX, 5.0);
                        glPointSize(5.0);
                        glDisable( GL_LIGHTING );
                        glColor3f(1,1,1);
                        glVertexPointer( 3, GL_DOUBLE, 0, pts);
                        glEnableClientState(GL_VERTEX_ARRAY);
                        glDrawArrays(GL_POINTS, 0, NumberOfVertices);
                        glDisableClientState(GL_VERTEX_ARRAY);
                }
        }
}
```

Get the Field

Get the data in this field
Fiber type is “point”

OpenGL: Make points 5 pixel in diameter, no lighting, constant white

OpenGL: Load vertex coordinates on graphics card and render as points


vish
File Options Inputs Outputs Help
1 1s 30
Network Scripting Tree Story
CameraLens
WebVish
Particles
theGrid
gridname GalaxyStars
TheTime
BBox
Render time: 0s
theField
Field available at T=0.39
fieldname Positions
myPointRender
inputfield theField.field
Viewer1 Viewer2 Viewer3 Viewer4
RAM Cache: 0 MB of ?
T=0.3995 ∈ [0, 7.99] (step 5 of 100, ΔT=0.0799)
Expand Left Cut Left less STOP more Cut Right Expand Right
Ready to load files or create objects.

Done!

HAPPY VISHING!

Homework D2 – OCT 27th

- Combine all of the above:
  - Make movie (10%)
  - Make point size a steerable parameter (15%)
  - Point render object needs to set bounding box (20%)
  - Make point colors a context-relative steerable parameter, provide screenshot (25%)
  - Create a Fish<Field> object that computes the norm of the overall acceleration (type *tvector*) and exports it as "double" value (25%)
    - double a_total = $\sum |a_i|$ where a_i is available as a field on the Particles.f5
  - Steer point render size with overall acceleration (5%)
- Optional:
  - Implement a render object that demonstrates an aspect of Geometric Algebra such as wedge product, rotation (+50%)
  - Hint: PhysicalSpace provides point, tvector, bivector types