Universal Pointer Implementation Notes

August 5, 2008

Morgan McGuire
Williams College

1 Introduction

A Universal Pointer allows indirection of any language feature that accesses a value. It is an adapter that allows a variable, a reference, a pointer, a garbage collected object, an overloaded operator, a pair of functions, or a pair of methods (and all of the const/non-const variations in C++) to all act like a simple pointer. One can even use recursive universal pointers, mimicking the chain-reactions of functional reactive programming languages. The latter is not a technique that we’ve found to be very practical, though, because the cascades are hard to control and debug.

We introduce universal pointers and use them to bolt new GUIs onto existing code. Because universal pointers adapt to any means of communicating a value, they impose little restriction on program state and allow a single programmatic interface for the GUI. Because they can both read and write state, they facilitate bidirectional synchronization.

Previous similar approaches required new languages and language features. We show that for OOP languages, universal pointers are just a data structure, and do not require new languages or compiler features. We make heavy use of overloading and templates in C++ and reflection in Java to implement them. Similar techniques are possible in other languages like Lua, Python, and C#.

We’ve released a skinnable OpenGL GUI based on universal pointers as open source in the G3D project (http://g3d-cpp.sf.net). This is appropriate for use in games, student assignments, and graphics research, all cases where an easy-to-use GUI that is tightly integrated with 3D rendering is desirable. See the Pointer and GuiPane classes in that library for the preferred implementation.

2 Syntax

Here are examples of creating, reading, and writing with universal pointers in C++. The syntax looks almost identical to that of regular pointers:

```c++
int get();
void set(int& x);
float f = 0.0f;
std::string s = "hello";

Pointer<int> iPtr (&get, &set);
Pointer<float> fPtr( &f );
Pointer<std::string> sPtr = &s;

printf("%d", *iPtr);
*iPtr = 1;
*sPtr += *iPtr;
std::string x = s->substr(3, 2);
```

3 Application to GUIs

You don’t normally create universal pointers explicitly. The idea is that they are adapters built into the GUI API, which are not seen in normal programming practice. For example, the code:

```c++
std::string filename;
pane->textBox("Filename", &filename);
```

appears to just pass a pointer from filename to the text box factory method. Internally that method converts the real C++ pointer into a universal pointer. The remainder of the implementation of the text box (which is written once inside the GUI API, not per-program) uses the universal pointer as if it were a C++ pointer. The distinction is that the same text box implementation can work with:

```c++
Level* level = new Level();
pane->textBox("Filename", level, &Level::getFile, &Level::setFile);
```

and even:

```c++
boost::shared_ptr< Level > level = new Level();
pane->textBox("Filename", level, &Level::getFile, &Level::setFile);
```

In a GUI, the universal pointer removes the need for a callback that sets the value in the program. This eliminates boilerplate code, which encourages programmers to use ad hoc GUIs for rapid debugging and reduces the cost of writing and maintaining production GUIs. For a quick (albeit admittedly biased and anecdotal) comparison, I looked at code from previous real-world consulting projects and compared the number of statements required to add a check box or labeled text box bound to a pre-existing variable. The results are shown in table ??.

<table>
<thead>
<tr>
<th>API</th>
<th>Check Box</th>
<th>Text Box</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ wxWidgets</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Typical Custom* Game GUI</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>Java Swing</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>C# WPF</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Universal Pointer GUI</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Sample programming statements to add a bound control.
* In the Titan Quest engine by Iron Lore Entertainment, published by THQ

One could argue the numbers in table ?? slightly either way, but other APIs clearly have more than four times the overhead of our approach. Section ?? shows two examples of the kind of code used by other APIs.

4 Implementations

These are minimalist implementations that demonstrate the techniques for implementing universal pointers in C++ and Java. See our website or the proceedings CD for practical implementations that have comments and more features. In the Java implementation, exception handlers have been removed for brevity.
4.1 C++

template<class T> class Pointer {
    class Interface {
        public:
            virtual ~Interface() {}
            virtual T get() const = 0;
            virtual Interface* clone() const = 0;
    }

class Memory : public Interface {
    T* value;
    public:
        Memory(T* value) : value(value) { }
        void set(T v) { *value = v; }
        T get() const { return *value; }
        Interface* clone() const { return new Memory(value); }
    }

template<class Class, typename Getter, typename Setter>
    class Accessor : public Interface {
        Class* obj;
        Getter getter;
        Setter setter;
        public:
            Accessor(Class* obj, Getter getter, Setter setter) :
                obj(obj), getter(getter), setter(setter) {}
            void set(T v) { (obj->*setter)(v); }
            T get() const { return (obj->*getter)(); }
            Interface* clone() const { return new Accessor(obj, getter, setter); }
    }

    Interface* interface;
    public:
        Pointer() : interface(NULL) {};
        Pointer(T* v) : interface(new Memory(v)) {};
        Pointer(const Pointer& r) { delete interface;
            if (r.interface) interface = r.interface->clone();
            this[0] = p; }
        Pointer(const Pointer& p) : interface(NULL) { this[0] = p; }

        Pointer(const Pointer& p) : interface(NULL) { this[0] = p; }

template<class Class>
    Pointer(Class* obj, const T& (Class::*getter)() const, void (Class::*setter)(const T&)) :
        interface(new Accessor<Class, const T& (Class::*getter)() const, void (Class::*setter)(const T&)>obj, getter, setter)) {}

    // Not shown: overloaded constructors to handle all const
    // variations and reference counted pointers (e.g., shared_ptr)
    ~Pointer() { delete interface; }

    inline const T getValue() const { return interface->get(); }
    inline void setValue(const T& v) { interface->set(v); }
}

/* Supports non-const dereference operator */
    class Indirect {
        friend class Pointer;
        Pointer* p;
        Indirect(Pointer* p) : p(p) {}
        public:
            void operator=(const T& v) { p->setValue(v); }
            operator ValueType() const { return p->getValue(); }
        }

        inline Indirect operator*() { return Indirect(this); }
        inline const T operator*() const { return getValue(); }
    }

4.2 Java

    public class Pointer<T> {
        protected Object object;
        protected Field field;
        protected Method getter, setter

        public Pointer(Object obj, String set, String get) {
            object = obj;
            getter = obj.getClass().getMethod(get);
            Class[] argTypes = {getter.getReturnType()};
            setter = obj.getClass().getMethod(set, argTypes);
        }

        public Pointer(Object obj, String fieldName) {
            object = obj;
            field = obj.getClass().getField(fieldName);
        }

        public void setValue(T newValue) {
            if (field == null) setter.invoke(object, newValue);
            else field.set(object, newValue);
        }

        public ValueType getValue() {
            if (field == null) return (T)getter.invoke(object);
            else return (T)field.get(object);
        }
    }

5 Examples of Other APIs

Here are a Java and C++ example of what binding a new check box
to a pre-existing boolean ("var") looks like in other GUI APIs. All
of the code overhead comes from the need to create event handlers
that perform the data binding.

5.1 wxWidgets

    Header:

        enum { ... CHECK ...};
        wxCheckBox* check;
        void OnCheckEvent(wxCommandEvent& e);
        void SetCheck(bool b);

    Implementation:

        EVT_CHECKBOX(CHECK, GUI::OnCheckEvent)
        check = new wxCheckBox(parent, wxID_ANY, "Check");
        ...
        void GUI::OnCheckEvent(wxCommandEvent& e) {
            var = check->GetValue();
        }
void GUI::SetCheck(bool b) {
    var = b;
    check->SetValue(b);
}

5.2 Swing

JCheckBox check;
...
check = new JCheckBox("Check");
check.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
        var = check.isSelected();
    }
});
getContentPane.add(check);
...

void setCheck(boolean b) {
    var = b;
    check.setSelected(b);
}