HIGH LEVEL 3D OBJECT SELECTION FOR THE UNICON LANGUAGE

Jafar Al-Gharaibeh and Clinton Jeffery

University of Idaho
Computer Science Department
Moscow, Idaho 83843, USA
jafara@vandals.uidaho.edu
jeffery@cs.uidaho.edu

Iyad Abu Doush

Yarmouk University
Computer Science Department
Irbid, Jordan
Iyad.doush@yu.edu.jo

ABSTRACT

Most computer graphics applications depend heavily on user input. Within many games and virtual environments, for example, user input is essential to create and/or direct actions within a virtual world. Much of this input comes through direct interaction with the virtual world's content, usually using a mouse. Most programming languages' graphics libraries provide low level APIs for 3D object selection, which makes the process of building interactive 3D graphics applications a challenge for programmers, especially beginners. This paper describes a high level 3D object selection model which greatly simplifies that process. The model is realized within a very high level programming language called Unicon. In this work, Unicon's high level API for 3D graphics programming is extended to support 3D object selection with very minimal addition to the language level so most of the implementation details are hidden from the programmer. The model adds a layer of abstraction to 3D object selection making it similar to common GUI environments, where the programmer doesn't have to worry about how to make object selection work. The programmer writes event handlers and assigns them to 3D objects in the scene; the language automatically calls these handlers when an interaction happens with these objects.

Keywords: 3D Object Selection, Picking, 3D Interface, Language Design, Unicon.

1. INTRODUCTION

Most 3D applications such as collaborative virtual environments and games are interactive in nature. The user clicks on the 3D scene and picks objects. 3D selection plays a main role in some applications so that it is impossible to build such applications, without 3D selection support. Despite its importance, writing 3D selection code is not always easy. In OpenGL for example, it involves a lot of function calls and low level programming.

The Unicon language has a powerful set of built-in 3D facilities that help programmers rapidly develop 3D applications [1], but it did not initially support 3D object selection. The relatively complicated and low-level object selection mechanisms used in OpenGL were inappropriate for a very high level language. Building the 3D selection mechanism into the VM runtime system makes it possible to hide all of the low level semantics from programs written in Unicon programming language. This paper describes the design, implementation and use of high level 3D selection facilities that were introduced to the Unicon language.

Different programming languages and graphics libraries have various mechanisms to implement 3D object selection. These include, but are not limited to:

- Color-coding
- Ray tracing
- Special rendering modes.

Color-coding involves rendering each primitive in a unique color in an off-screen window buffer so that the user does not notice this process, and then reading the pixel under the current cursor location. The color value of the pixel determines the primitive that the user selected. This technique provides good performance, especially with a small number of selectable objects. The drawbacks of this technique are:

- There is no depth information with the selected object.
- Only the closest objects to the cursor (camera) can be selected. Objects that hide under other objects cannot be selected.
- The system cannot support more selectable objects than there are unique colors. This limitation has grown less onerous over time since most systems support millions of colors.
- The unique color must be the same when it was rendered and when it was read back; lighting, dithering or any other setting that might affect how the system interprets the color could affect selection.

Ray tracing selection works by generating a pick ray from the mouse location to the far z-plane and testing if this ray intersects with objects in the scene.

Special rendering modes like the OpenGL selection mode work by making a separate rendering pass over the scene and restricting the viewing volume to a small area around the cursor. Objects that happen to fall within this viewing volume are considered to be selected.

2. RELATED WORK

Most of the programming languages that have 3D graphics APIs support mouse interactions and 3D object selection. Some of these languages provide very low level APIs that directly reflect functionalities in the underlying graphics libraries. A popular

example is the C API for OpenGL. In OpenGL it is up to the programmer to make numerous calls and prepare buffers to collect and process all of the results. Other languages hide some or all of these underlying implementation details to provide a higher level API at the language level.

2.1. OpenGL Selection Mode

OpenGL provides a mechanism for object selection through a special rendering mode. In selection rendering mode, instead of rendering the scene as color values to the color buffer, only "names" are rendered to a selection buffer. In this special mode the programmer supplies storage for results and sets up a special "pick" matrix with a view volume centered around the mouse cursor (Figure 1). Objects that are rendered into this selection

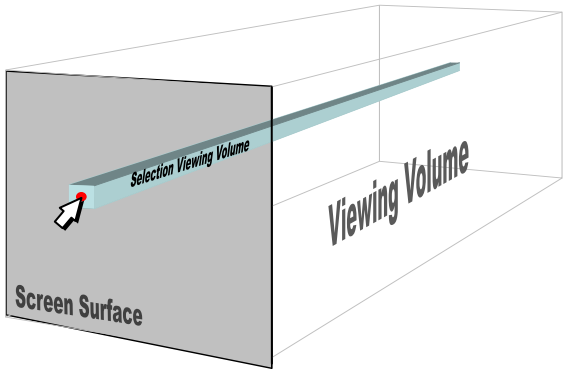

Figure 1. *Selection viewing volume centered around the mouse cursor.*

viewing volume are reported to the user as selection hits [2].
Picking is a special case of selection in which clipping is set up such that only a small region of the screen is visible: whatever is visible is then deemed to have been "picked". The programming steps are:

- Restrict rendering to a small region near pointer
- Use gluPickMatrix() on projection matrix
- Enter selection mode; re-render scene and give 3D objects unique integer identifiers
- Primitives drawn near cursor cause hits
- Exit selection; analyze hit records

When using OpenGL selection, the programmer has to work with 6 functions dedicated to selection plus several other functions that must be used with selection to make it work, for a total of 11 functions. In addition, the programmer must also setup a data structure to hold the selection results.

2.2. **Java3D API**

Java3D provides a rich class library for 3D graphics which include picking utility classes [3]. Two basic approaches can be used to achieve picking in Java3D: use objects of built-in special purpose picking classes, or use instances of more general custom picking classes. The picking package includes classes for pick/rotate, pick/translate, and pick/zoom. The user can press the mouse button over an object and drag to achieve one of the functionalities of rotate, translate or zoom based on the class type.

The picking classes can be setup to use different mouse buttons to give access to the different functionalities simultaneously [3]. The following steps summarize the process of making a selectable object in Java3D [3]:

- Create a scene graph
- Create a picking behavior object with root, canvas, and bounds specification
- Add the behavior object to the scene graph
- Enable the appropriate capabilities for scene graph objects

Adding user-defined actions requires creating custom behavior with custom picking classes and enabling/disabling many attributes for the required objects and their corresponding behavior objects. In summary, Java3D has a high level object selection mechanism compared to the C OpenGL API. But using it still requires a substantial amount of work from the programmer [4]. There are many packages and classes to keep track of and more attributes to remember and object capabilities to turn on or off. That is not to mention the inherent complexity in using the whole graphics library with more than 150 classes, which dictates how to setup the selectable objects, how to add them to a specific canvas and where to insert them in the scene graph before the whole selection process can be made possible.

3. LANGUAGE INTERFACE

3D graphics facilities in Unicon are designed to provide a simple programming interface. Instead of requiring programmers to learn hundreds of functions or methods in OpenGL or Java3D, Unicon, which is built on top of OpenGL, hides most of the underlying details yet preserves a lot of powerful 3D functionality. Basic functionality includes primitives, transformations, lighting, and texturing [5].
In 3D selection, the same principle was followed; hiding all of the unnecessary details and minimizing the language interface. In Unicon, one existing function was extended for selection, plus a keyword and a window attribute were introduced. A keyword in Unicon is a predefined global symbol, distinguished from ordinary variables by a leading ampersand, whose value is governed by the language control structures and built-in functions. Keywords are key components in both Icon and Unicon programming languages [6] [7].
Working with 3D selection in Unicon is easy and straight forward. The new keyword (&pick) provides access to 3D selection results. A new window attribute (pick) enables and disables 3D selection. The term "pick" was adopted to denote the 3D selection feature to avoid confusion with other uses of "select". The term "select" is heavily used in different contexts in programming languages and libraries. Clipboard contents, text regions and TCP sockets are examples of such use. The meaning of "pick" also conforms very well with its role in the language which is selecting or picking objects. A few steps are required to use 3D selection in Unicon. These steps are summarized by the following:

- Enable/disable the selection (on/off)
- Give selectable 3D objects unique string names
- Collect selection results for mouse input events through the keyword &pick

3.1. Controlling the selection state

Turning on/off 3D selection is controlled by the Unicon function WAttrib() [8]. WAttrib() is a generic routine for getting or setting a window's attributes. To turn on 3D selection at any point in the program, the following statement is inserted:

```
WAttrib("pick=on")
```

To turn off the 3D selection simply make another call to WAttrib() like the following:

```
WAttrib("pick=off")
```

To check the current 3D selection state (on or off) the function WAttrib() can be called with the string parameter "pick" (i.e WAttrib("pick")) and it will return a string value of "on" or "off" depending on the current 3D selection state. By default 3D selection is turned off. The program can turn on and off the 3D selection depending on the program requirements. For better performance it is recommended to turn off selection for any non-selectable object in the scene.

3.2. Naming 3D Objects

3D Objects are defined by their corresponding rendered primitives. The function WSection() [5] marks the beginning and the ending of a section that holds a 3D object. A call to the function WSection() with a parameter string marks the beginning of a 3D object with the string as its name. Another call to WSection() with no string parameter marks the end of the 3D objects. All of the rendered graphics between a beginning WSection() and its corresponding ending WSection() are parts of the same object. To be selectable, a 3D object must have at least one graphical primitive, such as a line or a sphere. The string name should be unique to distinguish different objects from each other. Different objects could have the same name if the same action would be taken no matter which of these objects is picked. The following code fragment is an example of named 3D objects. It simply draws a red rectangle and gives it the name "redrect".

```
WSection("redrect")          # beginning of object Fg("red")
FillPolygon(0,0,0, 0,1,0, 1,1,0, 1,0,0)
WSection()                   # end of the object
```

In the example above, the call WSection("redrect") marks the beginning of a new object with the name redrect. Fg("red")does not affect selection because it does not produce a rendered object. FillPolygon(0,0,0, 0,1,0, 1,1,0, 1,0,0)on the other hand does affect selection because it produces a rendered object, and it actually represents the object named redrect. WSection()marks the end of the object named redrect.

3.3. Retrieving Picked Objects

In general, picking objects is associated with the mouse. In Unicon, mouse events are tracked through a set of keywords. &lpress and &rpress for example denote values that indicate that there was a left click or right click event, respectively. The Unicon function Event() produces these events from the event queue. It also generates other information related to such events such as the x and y coordinates of the mouse cursor at the time of the click. &pick was designed to work in the same fashion with mouse clicks. If selection is enabled, &pick generates all of the string names of the objects under the cursor, one at a time. The following code fragment writes all of the objects' names that were picked by the mouse left-click:

```
every picked_object := &pick do
  write(" picked object :", picked_object)
```

If there were no selectable objects under the cursor at the time of the event, &pick just fails and produces no results. &pick gets its results from both left-clicks and right-clicks.

3.4. Complete Example

This section presents a simple full example program. The example demonstrates the use of the 3D selection mechanism in Unicon. Three spheres, red green and blue, are drawn in a 3D graphics window. The red and blue spheres are selectable but the green is not. The user can click on any place in the window and the program reports the picked object to the user. If the user clicks on the red or blue sphere he will get the message "you picked red ball" or "you picked blue ball". If the user clicked anywhere else including on the green ball he will get the message "you picked nothing". That is because the selection is off for the green ball so it is not selectable.

```
procedure main()
# open a 3D window and make it the default
&window := open("3D selection in Unicon",
                "gl","size=500,500")
# begin a new selectable section/object
WAttrib("pick=on")           #turn on 3D selection
WSection("red ball")
  Fg("red")
  DrawSphere(1, 0.5, 0, 0.5)
WSection() # end of the red ball

# Draw a nonselectable green ball
WAttrib("pick=off")          #turn off 3D selection
Fg("green")
DrawSphere(-1, 0.5, 0, 0.5)

# begin a new selectable section/object
WAttrib("pick=on")           #turn on 3D selection
WSection("blue ball")
   Fg("blue")
   DrawSphere(0, -0.5, 0, 0.5)
WSection() # end of the blue ball

#setup the eye to look at the spheres
Eye(0,0,4, 0,0,0, 0,1,0)
Refresh()

# enter an event loop to handle user events
repeat{
 case \Event() of {
   &lpress | &rpress : write("you picked : ", &pick | "nothing" )
   }
 }
end
```

4. EVENT-DRIVEN INTERFACE FOR 3D OBJECT SELECTION

Using 3D selection directly through WSection() function and keyword &pick is easy. The programmer gives objects unique names using WSection(), collects selected objects' names by scanning string names generated by &pick, and then takes the appropriate action based on the selected object. For small programs with few selectable objects this is a trivial task, but as the program and its number of selectable objects gets larger, managing selectable objects and the actions to be taken becomes challenging especially if the objects are hierarchical. When using selection, from a high level view, the programmer defines a selectable object and assigns an action to be taken when this object is selected. It can be thought of as a graphical user interface (GUI) object; which is an on-screen entity that respond to user events. When a programmer creates a button for example, he does not worry about the button name (except to make it readable and meaningful), and he does not worry about how or when this button was clicked. All the programmer cares about is to take an action if this button is clicked. This section discusses the introduction of a new class to the Unicon language for the purpose of managing 3D selection and adding a high level abstracted layer for using 3D selection.

4.1. Design for the 3D Selection Class

A new class was introduced with the name Selection3D. The class holds information about what objects are selectable, what events should these objects respond to and what is the action to be taken when an object is selected and receives an event that it should respond to. Any selectable object can respond to one or more mouse events. A separate table for each one of these mouse events (except for CLICK and DRAG, they simply reuse other events' tables) keeps all of the objects that can respond to that particular event. For example there is a table of all objects that can respond to a LEFT_CLICK event if any of these objects were selected. This table is called Tleft_click. Figure 2 shows the mapping from all of the mouse events that are recognized by this class to their corresponding tables.

The Selection3D class uses another helper class to store information about each selectable object: its name (given by the user), the action associated with it, the class objects that holds the action if the action is a method, and the event type (specified by the user). Figure 3 shows the two classes and their relationship.

CLICK	➔	Tleft_click and Tright_click
LEFT_CLICK	➔	Tleft_click
RIGHT_CLICK	➔	Tright_click
DOUBLE_CLICK	➔	Tdouble_click
DRAG	➔	Tleft_drag and Tright_drag
LEFT_DRAG	➔	Tleft_drag
RIGHT_DRAG	➔	Tright_drag

Figure 2. *Events that are recognized by the Selection3D class and the mapping between these events and their corresponding object tables.*

Figure 3. *UML diagram for the classes used to manage/control 3D object selection.*

4.2. Using the Selection3D class

To make the Selection3D class available for a program in Unicon, the following statement should be added at the beginning of the source code.

```
link Selection3D
```

An instance of this class can be created simply by an assignment statement. The following example line creates this instance and store it in a variable called select3D

```
select3D := Selection3D()
```

The Selection3D class has three methods that can be called to manage the 3D selection in the program. The first method is selectable(), which is used to register new 3D objects to make them selectable. This method takes up to four parameters in the following order

- A string name of the 3D object. The name does not affect the selection behavior in any way.
- A procedure/method name to be called when this 3D object is selected
- An optional event type which could be any of the event types shown in Figure 2. The default event type is LEFT_CLICK
- The class object that has the method name (second parameter). This is only valid (and mandatory) if the second parameter is a method which is part of a class object.

The method selectable() returns a string value which is referred to as selection_id. selection_id is a unique value that can then be passed to WSection() to mark a new 3D object name. The following is an example of such use:

```
select_id := select3D.selectable("red ball", on_red_ball)
WSection(select_id)
```

In this example a 3D object named "red ball" was registered to be selectable. The procedure on_red_ball will be called if this 3D object is selected. The third parameter is omitted which means the 3D object responds to the default event type and that is mouse left click. on_red_ball is assumed to be a procedure name (not a method) so nothing should be passed as a fourth parameter (omitted also) because a procedure is not part of any object unlike a method. The example also shows how is the returned value select_id was passed to WSection().

The second method in Selection3D class is add_action(). This method takes four parameters exactly like selectable() except for the first parameter. Instead of taking a random string name, it takes a string name that was returned and registered by selectable() to add another action or response to another kind of event. In the example above the following line of code can be added right after the first line to make the red ball respond to mouse right click by calling the procedure on_right_click():

```
select3D.add_action(select_id, on_right_click,
                    select3D.RIGHT_CLICK)
```

The third and the final method in the Selection3D class is handle_events(). Normally this method should be part of the event handling loop in the program. At least all types of mouse events in Unicon should be passed to this method. This lets the Selection3D class collect events information and picked objects through &pick and take the appropriate action. Failure to pass any kind of mouse events to the Selection3D class might cause it not to produce the intended behavior. A correct way to use the method handle_events() is shown toward the end of the example in the following section:

4.3. Complete example

```
link selection3D
global select3D

procedure on_red_ball()
  write(" You picked the red ball!")
end

procedure on_blue_ball()
  write("You picked the blue ball")
end

procedure main()
&window := open("3D selection in Unicon",
                "gl","size=500,500")
select3D := Selection3D()

# begin a new selectable section/object
WAttrib("pick=on")  #turn on 3D selection
select_id := select3D.selectable("red ball", on_red_ball)

WSection(select_id)
  Fg("red")
  DrawSphere(1, 0.5, 0, 0.5)
WSection() # end of the red ball

# Draw a nonselectable green ball
WAttrib("pick=off")  #turn off 3D selection
Fg("green")
DrawSphere(-1, 0.5, 0, 0.5)

# begin a new selectable section/object
WAttrib("pick=on")  #turn on 3D selection
select_id := select3D.selectable("blue ball", on_blue_ball)

WSection(select_id)
  Fg("blue")
  DrawSphere(0, -0.5, 0, 0.5)
WSection() # end of the blue ball

#setup the eye to look at the spheres
Eye(0,0,4, 0,0,0, 0,1,0)
Refresh()

# enter an event loop to handle user events
repeat{
 if (ev := \Event()) then{
   select3D.handle_event(ev)
   }
 }
end
```

5. IMPLEMENTATION DETAILS

3D selection in OpenGL requires rendering the scene in a special rendering mode called the GL_SELECT *rendering mode* [2]. In this mode, the scene should be redrawn in the off-screen buffer - without swapping it with the front buffer so that this process would be hidden from the user. After setting up the selection environment, objects in the scene must be rendered along with unique integer names assigned to each one of them. This selection mode works by restricting the drawing to a very small area on the screen around the mouse cursor and creating the viewing volume from that area back to the far Z-plane. All of the objects that are rendered in this viewing volume are reported as being selected.

In Unicon, each 3D graphics window has a Unicon list of lists and records to keep track of all objects in a 3D graphics scene. This list is maintained internally in Unicon's window structure when creating a window with "gl" parameter. Each entry in this list contains information about the function name and the parameters of that function. When a window needs to be redrawn, the window is cleared, all attributes are reset to the defaults, and the Unicon list of lists is traversed to redraw every object in the scene [9]. This implementation is very useful for 3D selection. Unlike the usual use of selection in OpenGL in languages such as C, where the programmer should keep a separate rendering function with selection data for selection rendering mode, in Unicon the same rendering function can be used and the selection code would be executed or skipped dynamically depending on whether the rendering pass is regular rendering mode or selection rendering mode. The key points in the implementation of 3D selection in Unicon include:

- Two variables were introduced to control the execution of the selection code
    - A state variable to turn on and off the selection in the application
    - A variable to control when to execute the selection code. Even if the selection is on, the selection code should be executed only once after each mouse click (left click or right click do the same)
- List of String names to be mapped with the integer ids (OpenGL names) for selection
- String names are controlled by WSection() calls which also stores the integer name mapped to each string name. i.e WSection() maintains both a string name and an integer name
- &pick generates all of the results from a list of string

6. CONCLUSIONS

This paper describes a very high level 3D object selection model that was created for the Unicon programming language, extending Unicon's 3D graphics API. The language hides most of the implementation details which leads to a design that puts fewer burdens on programmers when they use 3D object selection. The model also adds a layer of abstraction in the form of a class that encapsulates details and makes 3D object selection event-driven, similar to common GUI environments. Most programmers are familiar with GUI programming style, where the programmer doesn't have to worry about how to get the mouse to work or where the mouse pointer is at the time of a click. The programmer needs to worry only about what to do when an object is picked. He provides a handler and connects it to a specific object in the scene with a specific mouse event. This handler then is called whenever the user triggered that event.

This model was put to work in the CVE, an open source collaborative development environment (http://cve.sf.net). The model proved to be very effective and provided the CVE developers with a very easy access to 3D object selection, enabling them to add new functionalities to the CVE virtual world.

ACKNOWLEDMENT

The research was supported in part by NSF ATE grant number 0405072

7. REFERENCES

[1] Jeffery, C., El-khatib, O., Al-sharif, Z. & Martinez, N. (2005). *Programming Language Support For Collaborative Virtual Environments*. In Proceedings of 18th International Conference on Computer Animation and Social Agents. CGS.

[2] D. Shreiner, M. Woo, and J. Neider, *OpenGL(R) Programming Guide: The Official Guide to Learning* OpenGL, Version 1.2, 3rd ed. Addison-Wesley Longman, Amsterdam

[3] *The Java 3D Tutorial*, Sun Microsystems. Pages 4.1- 4.52. (Accessed Jan, 2011) http://java.sun.com/developer/onlineTraining/java3d/j3d_tutorial_ch4.pdf

[4] D. F. Savarese. *Learning to Fly*. *JAVAPro*, June 2003.

[5] Jeffery, C., Martinez, N., and Al-Gharaibeh, J. *Unicon 3D Graphics: User's Guide and Reference Manual* (November. 1, 2010), Unicon Technical Report #9b. (Accessed Jan, 2011) http://unicon.org/utr/utr9b.pdf

[6] R. Griswold, C. Jeffery, and G. Townsend. *Graphics Programming in Icon*. Peer to Peer Communications, San Jose CA, 1998.

[7] Jeffery, C., Mohamed, S., Parlett, R., Pereda, R. *Programming with Unicon*. (2005),. (Accessed Jan, 2011) http://unicon.org/book/ub.pdf.

[8] R. E. Griswold and M. T. Griswold. *The Icon Programming Language*. Peer to Peer Communications, San Jose, 1996.

[9] Jeffery, C., Martinez, N. *The Implementation of Graphics in Unicon Version 11* (June. 25, 2003), Unicon Technical Report #5a. (Accessed Jan, 2011) http://unicon.org/utr/utr5a.pdf