Subject: PickData Method on IDLgrWindow Object
Posted by Richard Tyc on Thu, 20 Apr 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Has anyone noticed any differences with the Pickdata method in the IDL 5.3.1 release.

I have a fairly complex window with one main model (oModel) which has several other Models added to it each of which having many objects (such as Polylines, texture-mapped polygons etc.) and each having a different transform.

Using the following statement on the overall model (allowing the user to pick anything on screen) :

```
pick = sState.oWindow->PickData(sState.oView,$
                                sState.oModel, $
                                [sEvent.x,sEvent.y],dataxyz)
```

The data point, dataxyz, never seems to look right. Even when I select a point on a IDLgrAxis object (which belongs directly to oModel), I never get the expected data point (ie. picking on the x axis should return a point with the range [0.0-1.0, 0 , 0] but there always seems to be some component of y
and z ?? It does however return correctly if I have selected the line (axis) or hit the background.

Anyone know whats going on or have similar experience ?

Rich

Subject: Re: PickData Method on IDLgrWindow Object
Posted by davidf on Thu, 11 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

Michael Plonski (mplonski@aer.com) writes:

> A while back, I was having a lot of difficulty understanding pickdata
> and select, since pickdata appeared to ignore the object (model) that
> you pass it. I didn't save the full thread of discussion with RSI but
> the key concept that was throwing me (quoted from the RSI response) was:
>
> "It is true that PickData DOES NOT check to see if the x,y is within the
> specified object. But that is precisely what Select is for."

Thank you, Mike, for this timely article.

Just the other day I was modifying my XImage program to show

folks how you could use the Pickdata function to select
an image value from the image object. (The updated program is on
my web page.) I got the program to work fairly quickly, but I
couldn't for the life of me figure out WHY it was working!

I would encourage everyone to save this article, because I'm
pretty sure you would have to be clairvoyant to capture this
information from the IDL documentation. In fact, the documentation
would lead you, probably, in exactly the opposite direction. :-(

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: PickData Method on IDLgrWindow Object
Posted by Michael Plonski on Thu, 11 May 2000 07:00:00 GMT
View Forum Message <> Reply to Message

You don't mention if things worked differntly in IDL 5.3.
A detailed description of pickdata from RSI follows.

A while back, I was having a lot of difficulty understanding pickdata
and select, since pickdata appeared to ignore the object (model) that
you pass it. I didn't save the full thread of discussion with RSI but
the key concept that was throwing me (quoted from the RSI response) was:

"It is true that PickData DOES NOT check to see if the x,y is within the
specified object. But that is precisely what Select is for."

This might be your problem, as I was getting what appeared to be
incorrect values from pickdata as well. If you want to see strange
things, overlay a surface above an image plane and then look
at pickdata results when the surface is a mesh vs solid. There was no
easy way to know with pickdata if you saw through the surface to the
image plane or hit a grid line on the mesh surface.

Hope this helps.

Mike Plonski

Subject:
Re: IDLgrWindow:Pickdata
Date:
Wed, 2 Feb 2000 14:57:55 -0700
From:
RSI E-mail Support <support@rsinc.com>
To:
'"mplonski@aer.com"' <mplonski@aer.com>

Mike:

Using Select first and then Pickdata is NOT a kluge. It is that way by design, which means both IDL and the external tools which IDL uses to render
graphics (such as OpenGL). The external tools have their own limitations,
which I am sure you can understand.

The Select operation (which means the OpenGL Select function call) is particularly expensive, since it performs an "invisible" rendering of an entire scene, creating a list of objects that are in the selection box.

The IDL Object Graphics designers chose, wisely, to separate the Select and
Pickdata operations, so that users would not be forced to pay the penalty of
the Select operation, when it is not really needed. The case where there is
only one object in a model would never require a Select operation; in this
case, you are ready to use Pickdata. It turns out to be fairly common for
complex scenes to have several models, where some of the models contain one
object. For this reason alone, it is important that the Pickdata method NOT
perform a Select operation "under-the-hood."

I have included, below, some comments from one of our Objects Graphics designers. I think you will find them very informative.

Sincerely,

Doug Loucks
Technical Support Engineer

Research Systems Incorporated
Phone: (303)413-3920
FAX: (303)786-9909
Email: support@rsinc.com
Please include my name on the Subject line of email replies. Thanks!

---------- Comments from one of our Object Graphics developers:

First, let me explain briefly how Select and PickData work.

Select() performs an "invisible" rendering of the entire scene, performing
hit testing within the selection box for each object in the scene. Objects
that are "drawn" into the selection box are returned in the object list.

PickData() simply reads the Z-buffer at the specified x,y to see if anything
at all got drawn at that location. If something was drawn there, the
following happens: PickData obtains the transform associated with the
object you specified in the call to PickData. It does not matter what
Select did the last time you called Select. (Select and PickData are
completely unrelated) It also does not matter where the x,y is. PickData
just obtains that object's user->screen transform and computes the
inverse-transpose of that transform. It then multiplies the specified x,y
by this inverse transform to obtain the corresponding data space coordinates
of the specified x,y. All it does is multiply the specified x,y by an
object's inverse transform if SOMETHING was drawn at that x,y. If x,y was
on some other object, the returned user space point will be *in terms of* or
relative to the object specified in the call to PickData. For example
suppose you have a sphere centered at (0,0,0) with radius=2 and a sphere
located at (5,5,5) with radius=2. If you call PickData with an x,y located
in the middle of the second sphere with the first sphere used as the
reference, you will get a returned user data coordinate of (5,5,?), instead
of the expected (0,0,?) because you used the first sphere's transform. Even
though the pick was made beyond the bounds of the first sphere.

We specify a "graphic" object, instead of a model, in PickData, because

each
graphic object in a given model can have its own unique transform, which
may
be different from the model's transform. Why? Each graphic object has
an
additional transform associated with it called the COORD_CONV transform,
specified by the XCOORD_CONV, YCOORD_CONV, and ZCOORD_CONV properties.
If
these properties are not specified, and left as their default values,
then
the graphic object's transform WILL be the same as the transform of the
model that contains that graphic object. But, in general, they are not
the
same. So, the suggestion of passing a model object in PickData instead
of a
graphic object does not work out well. Besides, the problem at hand
would
still exist if we used a model as the reference for the inverse
transform to
apply because there may be more than one model rendered at a given x,y.

There are several approaches to addressing the problem of associating
the
correct model with PickData.

1) Invoke the Select method every time before calling PickData. Use one
of
the objects returned by Select as the reference object for PickData. If
there are multiple objects returned by Select, you need to deal with
that
depending on the application. If there are no objects, there is no hit
and
no reason to call PickData. If you do call PickData, the result will be
in
terms of the object returned by Select.

If you want to call PickData again, for another x,y, then you really
need to
call Select again first, to make sure that you get the right object
associated with the new x,y. For example, if you are tracking the mouse
cursor and updating a "read-out" with object data values, then you
really
need to call Select and PickData for every mouse movement event. This
may
seem expensive, and it might be for very complex models. But for most
cases, Select is fast enough to remain interactive.

It is true that PickData DOES NOT check to see if the x,y is within the

specified object. But that is precisely what Select is for. Therefore, it
is not a kludge to call Select first; it is by design. The only thing that
PickData knows about the supplied object is its transform - PickData has no
idea where the object actually is on the screen.

Perhaps the "Pick" part of the PickData method was poorly named and is
partially responsible for the confusion. PickData does not perform a "pick"
as the PHIGS picking functions do. Perhaps you are thinking that PickData
is a full-fledged PHIGS pick - it is not. It just performs a vertex
multiply by a certain inverse transform, if the screen was dirty there.

So, why not just have PickData perform a "Select" inside of PickData???
Probably because the designers of this function thought that it was better
to give the user the choice and control over performing these two steps vs.
always doing them both. The obvious example is the case of a single object
- you don't need to call Select before PickData.

I think that this approach is the best, but here are some others.

2) Create (I don't mean draw one) a bounding box for each object. Call
PickData all you want without calling Select and see which bounding box the
data coordinate returned by PickData is inside of. This only works well
where the bounding box encloses the object with little or no extra space and
where objects don't interfere with each other. You would also have to be
careful about using COORD_CONV and which object you use for the reference
object in PickData.

3) Create a "mask" of the object you want PickData to be sensitive to in a
IDLgrBuffer by rendering just the one object into the buffer. If there is a
non-background color in the buffer at the specified x,y, then you have a hit
and can call PickData to get the values.
---------- end of comments ----------

Richard Tyc wrote:
>
> Has anyone noticed any differences with the Pickdata method in the IDL 5.3.1
> release.
>
> I have a fairly complex window with one main model (oModel) which has
> several other Models added to it each of which having many objects (such as
> Polylines, texture-mapped polygons etc.) and each having a different
> transform.
>
> Using the following statement on the overall model (allowing the user to
> pick anything on screen) :
> pick = sState.oWindow->PickData(sState.oView,$
> sState.oModel, $
> [sEvent.x,sEvent.y],dataxyz)
>
> The data point, dataxyz, never seems to look right. Even when I select a
> point on a IDLgrAxis object (which belongs directly to oModel), I never get
> the expected data point (ie. picking on the x axis should return a point
> with the range [0.0-1.0, 0 , 0] but there always seems to be some
> component of y
> and z ?? It does however return correctly if I have selected the line (axis)
> or hit the background.
>
> Anyone know whats going on or have similar experience ?
>
> Rich