

MASSIVE-3 / HIVEK Application Programming: Subjectivity

Chris Greenhalgh, 7th July 1999.

Contents:

- Visor
- Local Items
- Local Notification Events
- Subjective Rendering Characteristics

Visor

The standard user client(s) and the Renderer both support a process-local environment which is rendered over the top of the current view of the virtual world. This is typically used to hold personal navigation and interaction items, e.g. the compass, mouse vehicle, drop icon. This is never normally shared by another user (although this is technically possible).

Local Items

Items can be added to normal shared Environment which are not replicated, and are therefore only observable within the process which first created them. Unlike visor items, these are (subjectively) normal items within the shared environment, and can be anywhere in the item hierarchy.

An item is created local by setting the item flag `ITEM_LOCAL` when the item is created. For example (from `hiveClient.C`), the following creates a local Entity and a Geometry as its child:

```
Matrix3D transform;  
transform.setTranslate(0,1.5,0);  
ItemID nose = env->addNewEntity(&topId, &transform, NULL, NULL,  
                                LOCK_HARD, ITEM_LOCAL);  
env->addNewGeometry(&nose, "example_nose.dgl", LOCK_HARD);
```

Note that every child of a local item is also local automatically (if it was global then a remote parent would not know about its parent, and be unable to place it in the virtual world).

Items can only be made local when they are first created. Local items will always remain local and cannot be made global (the `ITEM_LOCAL` flag will be over-riden within the run-time system if these rules are broken).

Local Notification Events

Notification events, by default, are global in scope, i.e. they will be distributed to all replicas of an Environment. However, if the flag `localFlag` is set then the notification will only be visible within the process in which it originated. The following code fragment uses the final (optional) constructor argument to make the event local:

```
// SimpleEnvironment.C - (E.11) - An imaginary 'a'  
AgentID thisAgentID = env->getOwnerID();  
env->notification(new EventNotificationData("/user/keypress", 'a',  
                                           "a", thisAgentID,  
                                           nullAgent, nullItem,  
                                           nullItem, NULL, 1));
```

It will be reported to local EnvCallbacks in the normal manner.

Local notification could be used to coordinate multiple local scripts in their execution, without creating a flood of unnecessary network traffic.

Subjective Rendering Characteristics

The standard MASSIVE-3 renderer supports dynamic modification of the materials used to draw geometries. This change is specified by a `ViewCharacteristics` object, which has the following instance data:

```
float transparency; // View transform of materials
float brightness; // View transform of materials
float colourLevel; // 1 = normal colour, 0 = mono colour (below)
float monoColour[3]; // RGB of mono colour, default = white
```

where the fields have the following effects:

- A transparency of 0.0 has no effect, while a transparency of 1.0 would render the object wholly transparent (i.e. invisible).
- A brightness of 1.0 has no effect, while a brightness of 0.0 would render all materials as black, and a brightness of 2.0 would double the brightness of each RGB colour component (saturating at 1.0 for each component, independently).
- A colourLevel of 1.0 has no effect, while a colourLevel of 0.0 would cause all material colours to be mapped to different brightnesses of the specified monoColour. E.g. if monoColour is white (all components 1.0), and colourLevel is 0.0 then the object will be rendered black and white.
- monoColour is used with values of colourLevel less than 1.0, to specify the RGB colour to be used for materials.

`ViewCharacteristics` are entirely local to a given client program, and can be specified at two levels of granularity:

- Per-environment, applied to all geometries within that environment. This is specified in the `Locale` object used in replicating the `Environment`. At present the use of this facility is undefined. It is further modified by the `Renderer` itself, which currently maps awareness of an `Environment` to its brightness by default.
- Per-geometry, applied to an individual geometry. This is specified via a callback mechanism described below.

The following code fragment registers a per-geometry callback with the renderer. This callback is called immediately before rendering every `GeometryData` item in every rendering frame:

```
renderer.setItemViewCB(itemViewCB, this);
```

where `itemViewCB` is something like the following:

```
int HiveClient::itemViewCB(Renderer *renderer,
                           Environment *env,
                           ItemData *geomItem,
                           int selfFlag,
                           int &visibleFlag,
                           ViewCharacteristics &view,
                           void *userData)
{
    HiveClient *self = (HiveClient*)userData;

    // decide what to do with appearance of item . . . .
    view.colourLevel = 0.0;
    view.monoColour[0] = view.monoColour[1] = 1.0;
    view.monoColour[2] = 0.0;
    return 1;
}
```

This example draws all items in shades of yellow.

Note that the parameter `visibleFlag` can be cleared to prevent the item being rendered. This may already be zero on entry, e.g. for non-selected children of a `Switch` node.