



Introduction to WorldToolKit

Only way to learn is to do
and do
and do
and do
and do

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

1



WorldToolKit

- INTRODUCTION TO WORLDTOOLKIT
- THE UNIVERSE CLASS
- UNDERSTANDING SCENE GRAPHS
- NODES
- MOVABLE NODES
- NODE PROPERTIES
- GEOMETRY PROPERTIES
- MAKING OBJECTS MOVE
- WTK SENSORS
- MOTIONLINKS
- LIGHTING
- MATERIAL TABLES
- NODEPATHS
- TEXTURES
- PICKING GRAPHICAL OBJECTS

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

2



Introduction to WorldToolkit

- Portable, cross-platform software development system for building high-performance, real-time, integrated 3D applications for scientific and commercial use.
- > 900 high-level functions for configuring, interacting with, and controlling real-time simulations.
- WTK replaces the need to develop device drivers, file conversion filters, 3D functionality and other utilities with a well structured, easy to understand, and flexible API.



WTK intro

- WTK is *structured* in an object-oriented manner, although it *does not use inheritance or dynamic binding*.
- WTK functions are object-oriented in their naming conventions.
- They are grouped into classes. Classes include:
 - Universe (WTuniverse_*)
 - Geometry (Wtgeometry_*)
 - Viewpoints (WTviewpoint_*)
 - Sensors (Wtsensor_*)
 - Paths (WTpath_*)
 - Lights (WTnode_*)



Naming Conventions

- Each class of object has a typedef defining an object of that type:
 - *WTSensor* is a *sensor* object
 - *WTuniverse* is a *universe* object
 - *WTviewpoint* is a *viewpoint* object
- Objects are always dealt with through the use of pointers.
- State of an object is accessed through *get* and *set* functions.
- All methods of a given class have a naming convention that begins with the class name such as:
 - *WTuniverse_new*
 - *WTviewpoint_setposition*

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

5



"Sample" - Part I

```
#include "wt.h"           // include this in every WTK application

// Main function
void main(int argc, char *argv[]) {
    // Create a new default universe
    WTuniverse_new(WTDISPLAY_DEFAULT, WTWINDOW_DEFAULT);

    // Tie actionfn to the new universe
    // actionfn will be called for each frame during execution
    WTuniverse_setactions(actionfn);

    setupScene(); // Build the scene hierarchy
    setupSensor(); // Setup sensors (if any) to control the simulation

    WTwindow_zoomviewpoint(WTuniverse_getwindows()); // Zoom window to fit
    all objects

    WTuniverse_ready(); // Initiate WTK stuff

    WTuniverse_go();      // Starts simulation loop

    // Clean up when the loop stops (after WTuniverse_stop)
    WTuniverse_delete();
}
```

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

6



"Sample" - Actionfn

```
//-----  
// This function is called for each rendered frame  
static void actionfn(void)  
{  
    // Take care of the users keyboard actions  
    handleKeyPress( WTkeyboard_getkey());  
  
    // Insert any other calls to functions here  
}
```

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

7



"Sample" - HandleKeyPress

```
//-----  
// Called from the action function to handle users keyevents  
void handleKeyPress ( short key )  
{  
    switch(key) {  
  
        // Quit the simulation loop  
        case 'q':  
            WTuniverse_stop();  
            break;  
  
        // Zooms so that all the objects in the scene gets visible  
        case 'z':  
            WTwindow_zoomviewpoint(WTuniverse_getwindows());  
            break;  
  
        default:  
            break;  
    }  
}
```

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

8




"Sample" - SetupScene

```
//-----  
// Creates the graphical objects in the scene  
void setupScene( void ) {  
  
    WTnode *root;  
    Wtnode *node;  
  
    root = WTuniverse_getrootnodes();  
  
    // Add a point light to the scene positioned at origo (0, 0, 0)  
    WLightnode_newpoint(root);  
  
    // Load a vml-file containing a car  
    node = WNode_load(root, "car.wrl" , 1.0f);  
}
```

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

9



"Sample" - SetupSensor

```
//-----  
// Ties the mouse to the viewpoint to make it controllable  
void setupSensor ( void )  
{  
    WTsensord *mouse;  
    WTviewpoint *view;  
  
    // Create the mouse sensor  
    mouse = WMouse_new();  
    if (!mouse)  
        WError("Unable to create mousesensor");  
  
    // Get a pointer to the viewpoint  
    // This viewpoint is created by default by the WTuniverse_new function  
    view = WTuniverse_getviewpoint();  
    if (!view)  
        WError("Unable to pointer to first viewpoint");  
  
    // Attach the sensor to the viewpoint  
    WTviewpoint_addsensor(view, mouse);  
}
```

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

10

"Sample" - Result



2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

11

WTuniverse_new

- Must be the first WTK function call in a program; exceptions are the [WTinit](#) functions
- Can only be called once -- NEVER in the action function
- Performs the following functionality:
 - Initializes the graphics device
 - Configures for a particular output device
- Creates:
 - A default viewpoint
 - A default ambient light
 - A default root node
 - A window type

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

12



WTuniverse_delete

- Should be the last call in your main program
- Frees all of the objects in the universe.
- Including those that have been "removed" from the simulation
- Cleans up and closes the graphics hardware or WTK display



WTuniverse_ready

- Prepares the application for entry into the main loop
- Should be called **BEFORE** entering the simulation loop for the first time
- Should be called **AFTER** all graphical entities have been created.
- Assists in providing smooth frame-rates when objects come into view.
- For each new graphical entity added to the scene graph, WTuniverse_ready **MUST** be called.

WTuniverse_go

- Enters the simulation loop
- Loop is not exited until a call to `WTuniverse_stop` is made (usually from the universe action function).
- Must be called **AFTER** `WTuniverse_new` and `WTuniverse_ready`
- Is not reentrant. It **MUST NOT** be called from the universe action function

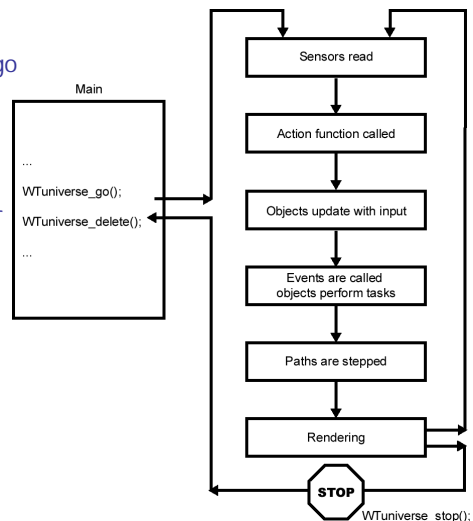
2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

15

Understanding the Simulation Manager

- Heart of all WTK applications
- Entered by calling `WTuniverse_go` or `WTuniverse_go1`
- Loop is broken with a call to `WTuniverse_stop`
- Order of events can be altered using `WTuniverse_seteventorder`



2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

16



Event order

- Change the event order so that objects are updated BEFORE any actions is taken.
- Especially useful for intersection testing as you want to update the objects positions *before* you test for intersection

```
// Create a new default universe
WTuniverse_new(WTDISPLAY_DEFAULT, WTWINDOW_DEFAULT);

short myevents[4];
myevents[0] = WTEVENT_OBJECTSENSOR;
myevents[1] = WTEVENT_TASKS;
myevents[2] = WTEVENT_PATHS;
myevents[3] = WTEVENT_ACTIONS;

WTuniverse_seteventorder(4, myevents);

WTuniverse_ready(); // Initiate WTK stuff
WTuniverse_go();    // Starts simulation loop
```

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

17



WTuniverse_setactions

- Used to define and control the activity in the simulation.
- Actions involving any WTK objects, graphical or otherwise, can be specified
- Is called by the simulation manager once each time through the simulation loop.
- Examples of actions which might occur in the action function:
 - Program termination
 - Simulation activities
 - Changes to rendering parameters
 - Handling sensor button input
 - Handling keyboard input
- Should be called **AFTER** `WTuniverse_new` and **BEFORE** `WTuniverse_ready`.

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

18

WTK coordinate systems

- Right hand axes - X, Y and Z
 - WorldToolKit uses a right-hand coordinate system. If the fingers of the right hand are pointed along the positive X axis, with the palm facing the positive Y axis, then the thumb points along the positive Z axis. This means that X is normally to the right, Z straight ahead, and Y points DOWN.
 - For rotations, if the thumb of the right hand points in the positive direction along an axis, then the fingers curl in the direction of positive rotation about that axis. Note that this applies to axes at any angle, not just the X, Y, and Z coordinate axes.

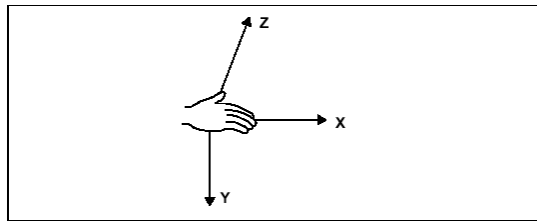


Figure: Illustration of right-hand rule

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

19

WTK Scene-graphs

- See lecture notes from Real-time graphics.
- Advantages of Scene Graphs
 - Object grouping
 - Level of detail switching
 - Instancing of geometry and entire scene graph's sub-trees, providing better memory usage
 - More powerful culling and more efficient database representation provides higher frame rates
 - Support for VRML and Open Inventor-style formats
 - Lights and environmental effects enabled only in parts of database
 - Multiple scenes

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

20



WTK Scene-graphs

- A number of methods exist for creating scene graphs.
 - Parts may be constructed using different methods
 - Function calls for:
 - Disassembling
 - Re-assembling
 - Re-arranging
 - The methods are:
 - Loading a scene graph description from a file. VRML 1.0/2.0, Open Inventor, and Multigen flt files contain hierarchically arranged data which correspond to WTK's scene graph.
 - Construct the graph node by node. WTK provides function for creating nodes and placing them at specific positions in the scene graph.
 - `WTnode_insertchild`, `WTnode_addchild`



WorldToolKit Node Types

- Nodes in WTK can be grouped into 3 distinct types
 - Geometry Nodes
 - Contain the representation of visible entities (vertices, polygons)
 - Node types include:
 - Geometry nodes
 - Movable Geometry nodes
 - 3D text
 - Attribute Nodes
 - Used to affect the way geometry nodes are rendered (the state).
 - Node types include:
 - Fog nodes
 - Light nodes
 - Movable Light nodes
 - Transform nodes

WorldToolKit Node Types

- Procedural Nodes
 - Used to control the way a scene graph is put together and processed.
 - Node types include:
 - Anchor nodes
 - Group nodes
 - Inline nodes
 - LOD nodes
 - Movable LOD nodes
 - Switch nodes
 - Movable Switch nodes
 - Separator nodes
 - Movable Separator nodes
 - Transform Separator nodes

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

23

Scene Graph State

- The state of the scene graph can be described as how your geometry is being rendered at any particular point in the scene graph.

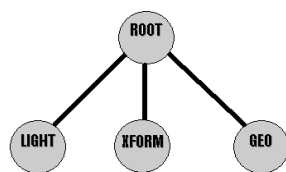


Figure 1

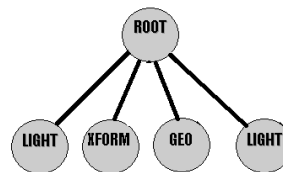


Figure 2

- In **figure 1** the Light Node would light the geometry in the GEO node since its state is pushed onto the stack before the geometry node is traversed. The transform node would also affect the state of the geometry since it precedes it in the scene graph.
- In **figure 2** a light node is added at the end of the scene graph. This node WILL NOT affect the state of the geometry since it is traversed after the geometry node; therefore the geometry node will not be affected by this light.

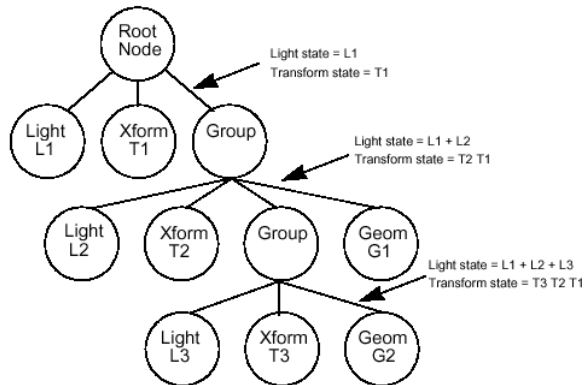
2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

24

State accumulation

- Because the current transformation state at some point in a scene graph is a concatenation of the transform nodes processed up to that point, and the current lighting state includes all the lights activated by processing light nodes up to that point, you can say that transform and light state “accumulate” as the scene graph tree is traversed, as shown in figure:



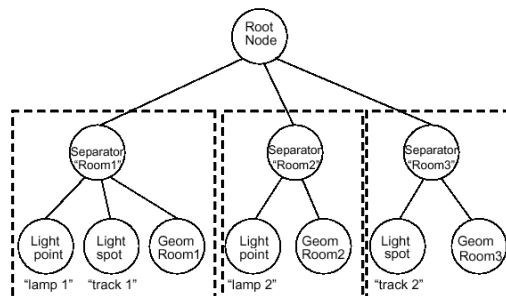
2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

25

Managing the state

- Separator and Transform Separator Nodes are used to manage the state of the scene graph by isolating the effects of the attribute nodes.
- Note that neither actively modifies the state of the scene graph, they prevent the descendant attribute nodes from affecting the state of the sibling nodes.
- The figure shows the use of the *Separator* nodes when localizing the affect of lights.



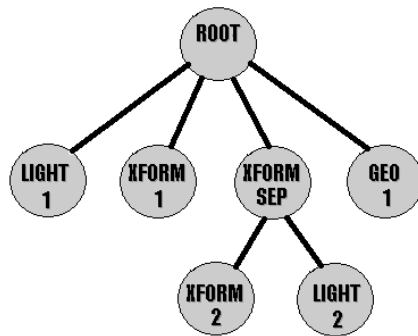
2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

26

Managing the state

- Transform separators** are used to localize the affect of transformations. The Light 2 in figure *will* affect (Geo 1). But the transform(Xform 2) won't.



2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

27

Node types

Node	Definition	Can it have children?	Does it effect state?
Geometry	Displays a set of polygons, together with a material	NO	NO
Fog	Simulates fog, smoke, or mist	NO	YES
Light	Specifies a light (point, directed, or spot)	NO	YES
Transform	Sets position and/or orientation information	NO	YES
Anchor	Contains a string property. Used of URL's for non-geometric information	YES	NO
Group	Has children by no other properties	YES	NO
Inline	Children are read in from a file	YES	NO

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

28



Node types

Node	Definition	Can it have children?	Does it effect state?
Level of Detail (LOD)	Swaps in objects as a function of the viewpoint distance	YES	NO
Root	Acts as the topmost node in a scene graph. Each scene graph can have only one root node. This node CANNOT be shared with any other graph. As the top node in its hierarchy, this node can have no parent node	YES	NO
Separator	Prevents state information from propagating from its descendant nodes to its sibling nodes	YES	NO
Switch	Controls which of its children will be traversed	YES	NO
Transform Separator	Prevents just the transform state from propagating from its descendant node to its sibling node	YES	NO

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

29



Finding individual nodes

- By using `WTnode_setname` a node can be given an individual name.
- By using `WTuniverse_findnodebyname` the node is located in the scene-graph.
- Example:

```
WTnode *test = WTnode_load(root, "test.3ds", 1.0f);
WTnode_setname(test, "TESTNODE");
WTnode *test2 = WTuniverse_findnodebyname("TESTNODE", 0);
```

- After this is: `test == test2`

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

30

Creating primitives

- When creating primitives it is important to note that all primitives are constructed using their local coordinate system, (0,0,0) being the center or midpoint of the geometry.
- Therefore when the geometry is inserted into the scene graph it is inserted at the World center (0,0,0).
- WTK provides functions for creating the following primitive types:

Type of Primitive	Function	Explanation
BLOCK	WTgeometry_newblock	creates a rectangular box
CYLINDER	WTgeometry_newcylinder	creates a cylinder
CONE	WTgeometry_newcone	creates a cone
SPHERE	WTgeometry_newsphere	creates a sphere
HEMISPHERE	WTgeometry_newhemisphere	creates the top half of a sphere
RECTANGLE	WTgeometry_newrectangle	creates a single rectangle
TRUNCATED CONE	WTgeometry_newtruncone	creates a truncated cone
EXTRUSION	WTgeometry_newextrusion	extrudes a contour into a geometry
TEXT3D	WTgeometry_newtext3d	creates a geometry representing a character (text) string

NOTE: When a primitive is created it is assigned a default material of matte white.

2000-11-10

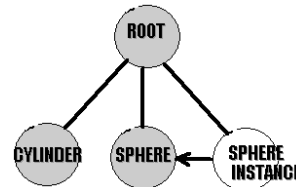
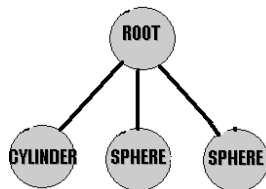
© Anders Backman, Dept. Computing Science, VR00 - WTK

31

Node Instances

- WTK allows the user to create an efficient database through the use of "instances" of nodes instead of complete copies.
- By creating multiple references to a single node instead of creating copies of it, memory usage is greatly reduced.

```
root = WTuniverse_getrootnodes();
WTgeometrynode_new(root, WTgeometry_newcylinder(2, 1, 10, FALSE, TRUE));
WTgeometrynode_new(root, WTgeometry_newsphere(1, 10, 10, FALSE, TRUE));
WTgeometrynode_new(root, WTgeometry_newsphere(1, 10, 10, FALSE, TRUE));
```



```
Root = WTuniverse_getrootnodes();
WTgeometrynode_new(root, WTgeometry_newcylinder(2, 1, 10,
FALSE, TRUE));
Wtnode *sphere = WTgeometrynode_new(root,
WTgeometry_newsphere(1, 10, 10, FALSE, TRUE);
Wtnode_addchild(root, sphere);
```

2000-11-10

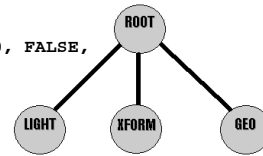
© Anders Backman, Dept. Computing Science, VR00 - WTK

32

Transformations

- `WTxformnode_new`
 - Syntax:
 - `WTnode *WTxformnode_new(WTnode *parent);`
 - Arguments:
 - **parent** -- The node will be inserted as the last child of the parent specified. If NULL is specified as the parent argument then the node is created as an "orphan" and can be added or inserted later.
 - Return Type:
 - A pointer to a `WTnode` object containing the new transform node.

```
WTnode *root, *node, *transform;
root = WTuniverse_getrootnodes();
WTpointlight_new(root);
WTxformnode_new(root);
WTgeometrynode_new(root, WTgeometry_newcylinder(2, 1, 10, FALSE,
        TRUE));
```



2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

33

Setting the Contents of the Transform Node

- Once the transform node is created its contents must be set.
- Transformations in WTK are stored in 4x4 matrices that specify the rotation and translation that will be applied to the state of the scene graph.
- There are functions that allow the user to set the absolute position and orientation of transform nodes.
- The following chart lists these functions and gives a brief explanation of their use.

FUNCTION	EXPLANATION
<code>WTnode_settranslation</code>	Sets only the absolute translation value of the transformation matrix.
<code>WTnode_setrotation</code>	Sets only the absolute rotation values of the transformation matrix. The rotation is specified in a 3x3 matrix
<code>WTnode_setorientation</code>	Sets the orientation (rotation) of the transformation matrix. The orientation is specified as a quaternion.
<code>WTnode_settransform</code>	Sets the translation and rotation of the transformation matrix. Values are specified in a 4x4 matrix.

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

34

Using a Separator Node

- **WTsepnode_new**
 - Creates a new separator node and adds it as the last child of the parent specified. If NULL is specified as the parent argument then the node is created as an orphan that can be added or inserted later.
 - Syntax:
 - WTnode * WTsepnode_new(WTnode * parent)
 - Arguments:
 - parent -- The parent node
 - Return Type:
 - A pointer to a WTnode object containing the separator node.

- **WTxformsepnode_new**
 - Creates a new transform separator and adds it as the last child of the parent specified. If NULL is specified as the parent argument then the node is created as an orphan that can be added or inserted later.
 - Syntax:
 - WTnode * WTxformsepnode_new(WTnode * parent);
 - Arguments:
 - parent -- The parent node
 - Return Type:
 - A pointer to a WTnode object containing the transform separator.

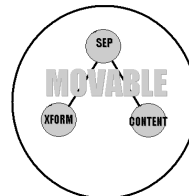
2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

35

Movable nodes

- There is a special kind of node called **Movable** that streamlines the process of creating objects, moving them with transform nodes and isolating them from each other with separator nodes.
- A movable contains a **separator** node or **transform separator** node, a **transform** node, and a **content** node.



Function	Type of Content Node	Type of Separator Node
WTmovgeometrynode_new	Geometry	Separator
WTmovlightnode_newpoint	Point Light	Transform Separator
WTmovlightnode_newdirected	Directed Light	Transform Separator
WTmovlightnode_newspot	Spot Light	Transform Separator
WTmovsepnode_new	Separator	Separator
WTmovswitchnode_new	Switch	Separator
WTmovlodnode_new	LOD	Separator
WTmovnode_load	Depends upon the file read	Depends on the file read

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

36



Instantiating a Movable Node

- In order to make an instance of a movable node, the function `WTmovnode_instance` is provided.
- Simply using `WTnode_addchild` or `WTnode_insertchild` isn't sufficient for use with movable nodes *since all instances would share the identical transformation component*.
- As a result, all instanced nodes would have the same position, translation and orientation.
- `WTmovnode_instance` creates a separate transform component for each movable node instance while still sharing data for the contents.
- In other words, *all information is shared except for the transform*.



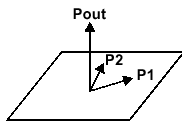
WTK Math

- The WTK math library contains functions for managing position and orientation data.
- The types used in the library is:
 - **WTp2** - 2D Vector – array of 2 floating point values.
 - **WTp3** - 3D Vector – array of 3 floating point values.
 - **WTq** - Quaternion – array of 4 floating point values.
 - **WTpq** - Coordinate Frame – structure containing WTp3 and WTq.
 - **WTm3** - 3D Matrix – 3x3 array of floating point values.
 - **WTm4** - 4D Matrix – 4x4 array of floating point values.
- Orientations are always stored in quaternion form.
- There exists conversion functions from/to euler angles and transformation matrices.
- Some of the functions are actually MACROS, beware of calls like:

```
WTp3_mults(pos, f(pos)); // f(pos) will be evaluated three times
```

WTK Math

- Example of functions:
 - WTp2 (2D vector)
 - [MACRO] `WTp2_init(WTp2 p); //Sets p[X] = p[Y] = 0`
 - [MACRO] `WTp2_norm(WTp2 p); // Normalizes P`
 - WTp3 (3D vector)
 - [MACRO] `WTp3_invert(WTp3 pin, WTp3 pout);`
 - // Negates pin, places result in pout
 - `void WTp3_cross(WTp3 p1, WTp3 p2, WTp3 pout);`
 - // Calculates the cross product between p1 and p2



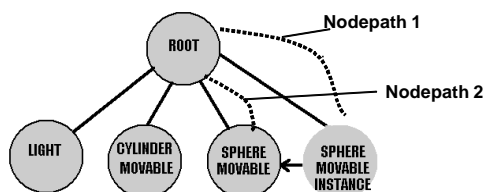
2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

39

Nodepaths

- As several nodes can share the same geometry, we need something that can separate the nodes apart.
- A nodepath is a mathematical entity that allows you to distinguish between multiple occurrences of the same node due to instancing. This allows you to indicate a specific occurrence of a node in the scene graph.
- There are two things you can do with nodepaths:
 - Perform intersection tests between specific nodepaths and other nodes in the scene graph.
 - Pick graphical entities rendered into a WTK window. The WTK picking functions generate the node-path of a picked geometry node.



2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

40



Creating Nodepaths

- **WTnodepath_new**
 - Use this function to create a new nodepath. A nodepath is fully specified by giving the bottom-most node of interest, the ancestor node, and the occurrence number.
 - SYNTAX:
 - `WTnodepath_new(WTnode *node, WTnode *ancestor, int which);`
 - ARGUMENTS:
 - `node` -- bottom-most node
 - `ancestor` -- The ancestor
 - `which` -- The occurrence of the node.
 - RETURN TYPE:
 - Returns a pointer to a new WTnodepath



Making objects move

- To make objects move, you have to write code in the action function. Or you can create tasks.
- Tasks is associated with a WTK object or a C structure.
- You set it up and WTK will call the task function once for each frame.
- **WTtask_new**
 - SYNTAX:
 - `WTtask *WTtask_new(void *objptr, WTtask_function fptr, float priority);`
 - ARGUMENTS:
 - `objptr` -- pointer to an object to which the function (fptr) will be associated
 - `fptr` -- Pointer to the task function.
 - `priority` -- Specifies the order in which the tasks will be called. Lower indicates higher priority.
 - RETURN TYPE:
 - Returns a pointer to a new WTtask



Tasks - Example

- For example, to add a task to a light, your application would include code similar to the following:

```
WNode *light;  
light = WLight_newspot(...);  
WTask_new(light, light_task, 2.5f);
```

where light_task is defined as follows:

```
void light_task(WNode *light) {  
    /* code that changes the light */  
    float v = WLightnode_getangle(light);  
    WLightnode_setangle(light, 2*v);  
}
```



What can we do in tasks?

- Anything can be done in a task or in the action function.
 - Movement - Translate, rotate objects using `Wnode_settranslation`, ...
 - Change in appearance - Move textures, change colors, ...
 - Testing for intersections - Collision detection, ...
 - Triggering other behavior - Play sounds, ...
 - Attaching a sensor

- Example:

```
case 'r':  
    WNode_settranslation(node, pos);  
    WSound_play(sound);  
    break;
```



Time/frame-based simulation

- **Framebased**
 - Lets say we want to move an object forward in x in a constant rate.
 - For each frame we could:
 - $p[X] = p[X] + \text{deltaX}$;
 - `WTtranslate(object, p)`
 - This would move the object `deltaX` units in the X direction for each time the scene has been rendered.
 - Usually this is NOT what we want. If the scene takes different time to render, depending on the viewpoints position and orientation (something complex gets into the view frustum) we will get different translation between different frames.
- Therefore we need to have time-based translation:
 - $\text{deltaX} = \text{deltaXperSecond} * \text{noSecondsSinceLastFrame}$;
 - $P[X] = p[X] + \text{deltaX}$
 - `WTtranslate(object, p)`

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

45



WTK sensors/Motionlinks

- **Sensors** are the devices by which we manipulate the simulation
 - The easiest to use: the mouse
 - Others: Spaceball, Insidettrak, ...
- **Motionlinks** connects objects that generates transformations (sensors and paths) to the objects we wish to manipulate: transforms, viewpoints, movables, nodepath
- Example:

```
Wtnode *transform = Wtxformnode_new(root);
Wtsensor *mouse = WTsensormouse_new();
Wtmotionlink *ml = Wtmotionlink_new(mouse, transform, WTSOURCE_SENSOR,
                                     WTTARGET_TRANSFORM);
```
- A mouse movement will affect the transformation.

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

46



Textures

- WTK supports the following texture formats:
 - Targa - .tga
 - SGI RGB format - .rgb and .rgba
 - JPEG – JFIF compliant - .jpg
- Usually objects are textured using a modeling tool.
- But there exists functions to texture geometry's "manually"
 - `WTgeometry_settexture`
 - `WTPoly_settexture`
 - `WTPoly_settextureuv`
 - `WTgeometry_settextureuv`
- And also for manipulating textures
 - `WTPoly_rotatetexture`
 - `WTPoly_scaletexture`
 - ...

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

47



Picking graphical objects

- `WTscreen_pickpoly`
 - This function obtains a pointer to the frontmost polygon rendered at the specified 2D screen point.
 - SYNTAX:
 - `WTPoly *WTscreen_pickpoly (int, screennum, WTP2 pt, Wtnodepath **nodepath, WTP3 p);`
 - ARGUMENTS:
 - `int screennum` – specifies a screen (`WTwindow_getscreen(WTuniverse_getwindows());`);
 - `WTP2 pt` – x, y in screen coordinates
 - `Wtnodepath **nodepath` – pointer to a pointer to a nodepath, filled with the nodepath that contains the picked polygon
 - `WTP3 p` – 3D point in world coordinates at which the selected polygon was intersected.
 - RETURN TYPE:
 - Returns a pointer to the picked polygon, null if no hit.
- The following example picks a polygon beneath the mouse and marks the node associated with that polygon with a bounding box.

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

48



Picking - Example

```
Void PickAndMarkWithBoundingBox( void )
{
    WTsensor *mouse = WTuniverse_getsensors();
    WTmouse_rawdata *raw = (WTmouse_rawdata *)WTsensor_getrawdata(mouse);

    // Get the position of the mouse in the window
    WTp2 mousePos = raw->pos[X], raw->pos[Y];

    WInodepath *nodepath=0;
    WInodepath pickedPoly=0;
    WInodepath p;
    WInode *pickedNode=0;
    // Pick the closest polygon under the screen coordinate mousePos
    pickedPoly = WInodepath_pickpoly(WInodepath_getscreen( WTuniverse_getwindows()),
                                     mousePos, &nodepath, p);
    if ( pickedPoly ) { // We hit something
        pickedNode = WInodepath_getnode( nodepath,
                                         WInodepath_numnodes( nodepath)-1 );
        WInodepath_delete( nodepath ); // Free the nodepath
        nodepath = 0;
    }
    WInode_boundingbox(pickedNode, !WInode_hasboundingbox(pickedNode));
}
```

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

49



Last words

- Look at the WTK tips page
 - Can be found from the <http://www.cs.umu.se/kurser/TDBD12/HT00/lab/wtk>
- Read all the material on the web-pages.
- Ask!
- Try/fail/try/success!!
- Look at demos

2000-11-10

© Anders Backman, Dept. Computing Science, VR00 - WTK

50