

CS 277 - Experimental Haptics

Lecture 3

“Open Source Framework CHAI 3D”



outline

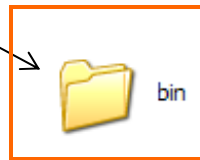
- organization of CHAI3D
- haptic devices - software interface
- creating a virtual world
- scene graph
- building object primitives
- programming haptic effects
- example
- developing your own primitives

outline

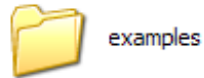
- organization of CHAI3D
- haptic devices - software interface
- creating a virtual world
- scene graph
- building object primitives
- programming haptic effects
- example
- developing your own primitives

organization

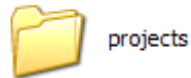
- executable files
DLL libraries



- source code for
all examples



- products from
compiling the CHAI3D
source code

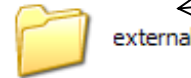


- project files

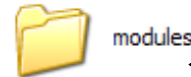
- HTML documentation



- third party libraries



- external module
(dynamic engines)

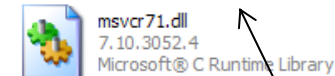
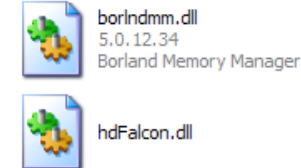
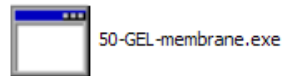
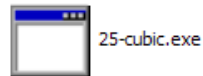
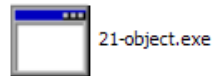
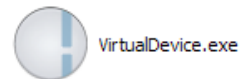
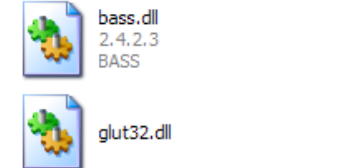
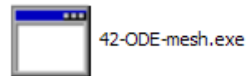
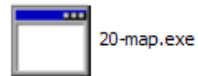
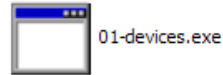
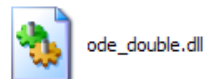


- source code

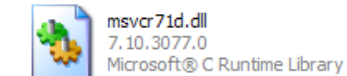
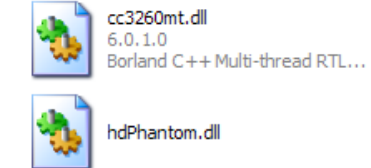
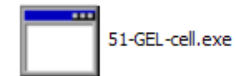
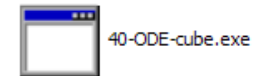
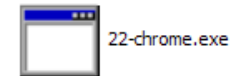
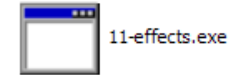


./bin

- data files (images, models, etc...)



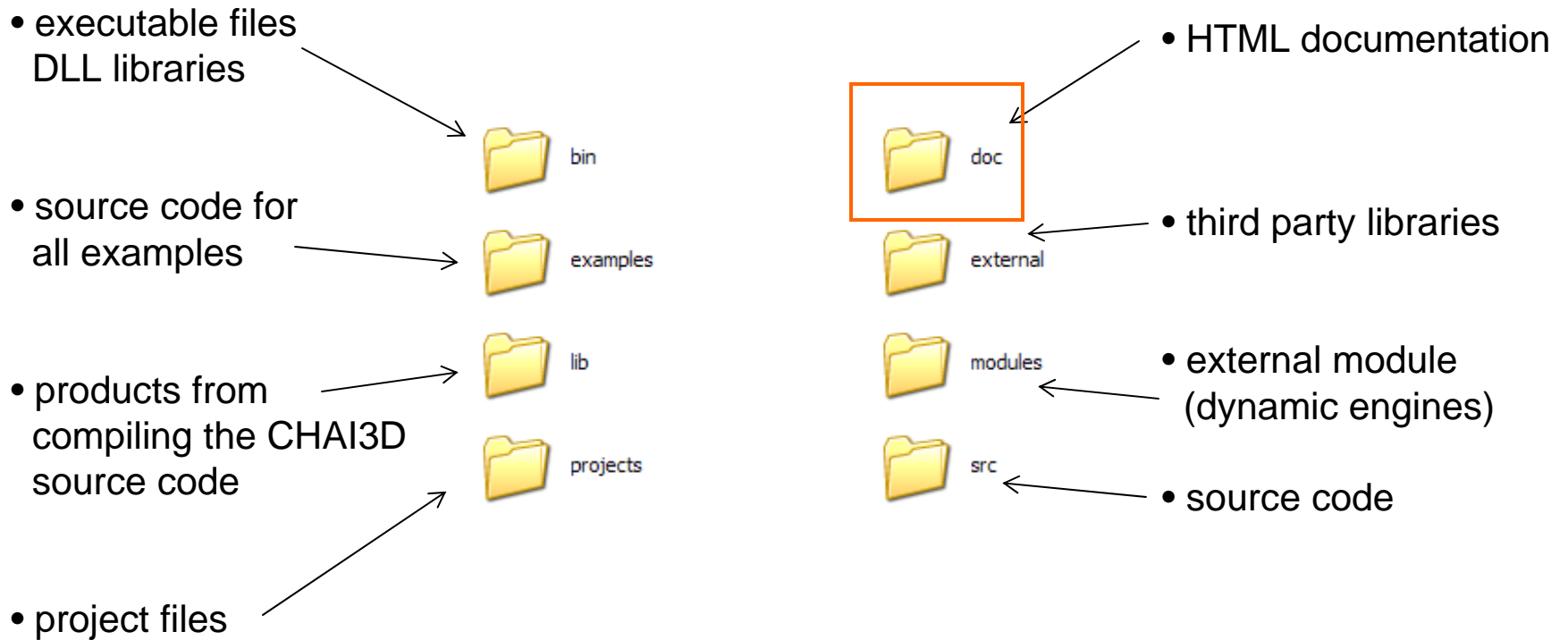
- examples (executables)



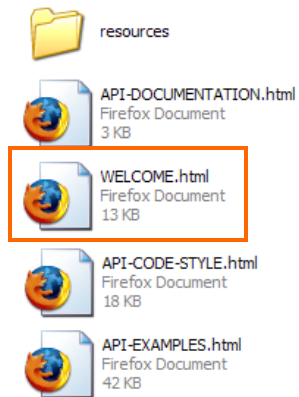
- virtual haptic device

- DLL files

organization



.doc



[Main Page](#) [Modules](#) [Classes](#) [Files](#)

CHAI3D Documentation

2.0.0

CHAI 3D is an open source set of C++ libraries for computer haptics, visualization and interactive real-time simulation. CHAI 3D supports several commercially-available three-, six- and seven-degree-of-freedom haptic devices, and makes it simple to support new custom force feedback devices.

For a general introduction to the CHAI 3D libraries, we highly recommend to review in numerical order the well documented examples provided with the framework.

For overviews of various selected features or general organization of CHAI 3D, see the above "**Modules**" and "**Files**" links.

Some functionalities or support for certain haptic devices may vary depending of the operating system you are using. CHAI 3D is currently supported on the following platforms:

Windows [32-bit]
[delta.x, omega.x, falcon, phantom, freedom6]

Windows [64-bit]
[delta.x, omega.x, phantom]

Mac OS-X [universal]
[delta.x, omega.x]

Linux [32-bit]
[delta.x, omega.x]

Linux [64-bit]
[delta.x, omega.x]

See <http://www.chai3d.org/> for more information about CHAI 3D.

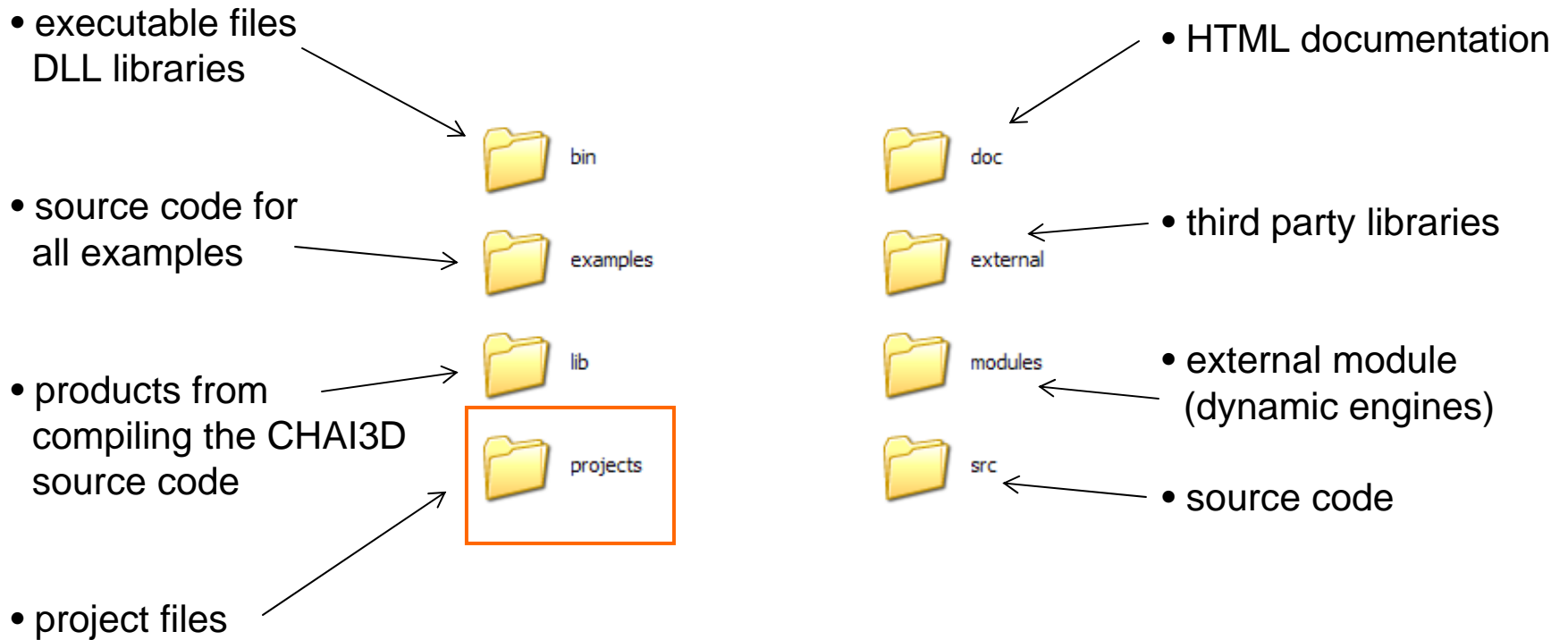
IMPORTANT NOTE: the online documentation for the CHAI 3D libraries is a continuous work-in-progress.

Although the large majority of classes have been documented properly, there might still be some poorly documented items. If you happen upon an undocumented or poorly documented class and / or class method which you find hard to understand, please give us a notice so we can rectify the situation.

CHAI3D 2.0.0 documentation
Please address any questions to support@chai3d.org
(C) 2003-2009 - CHAI 3D
All Rights Reserved.

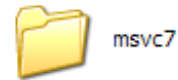
- HTML documentation

organization



./projects

- Borland Builder 6.0



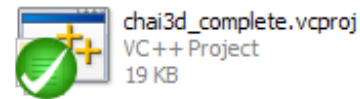
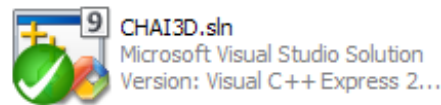
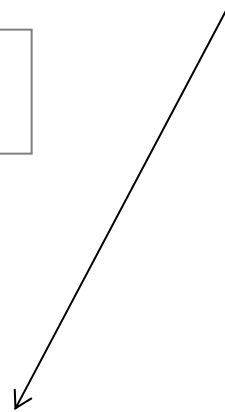
- MSVC 2003

- MSVC 2005



- MSVC 2008

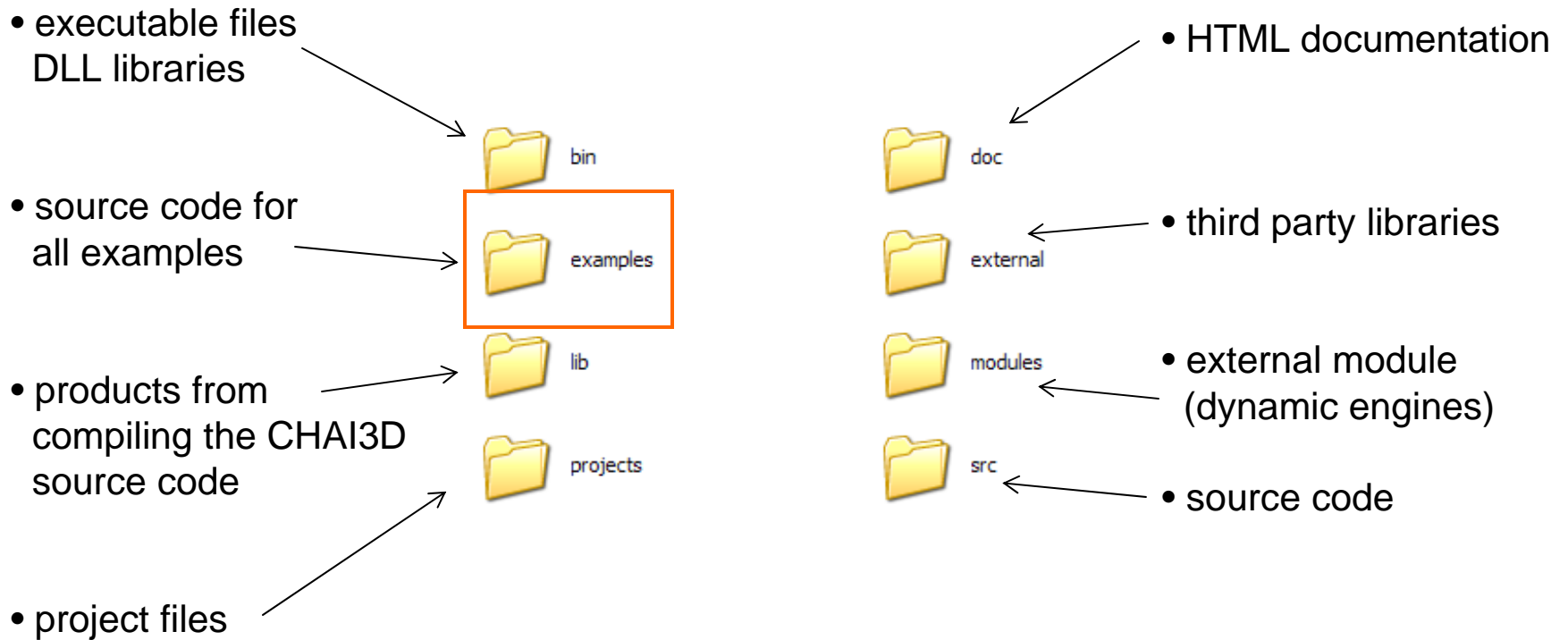
- MSVC : Microsoft Visual Studio X
- BBCP6 : Borland Builder C++ 6.0



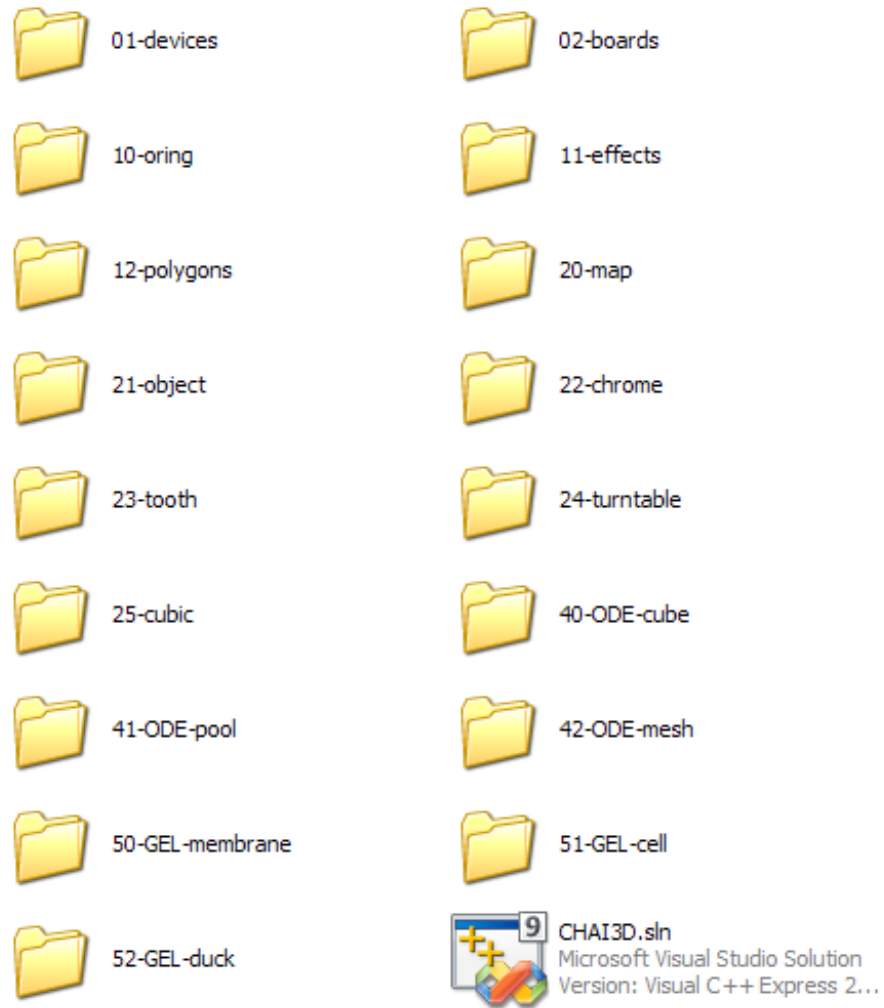
- Solution File (Open this one!)

- CHAI 3D Project File

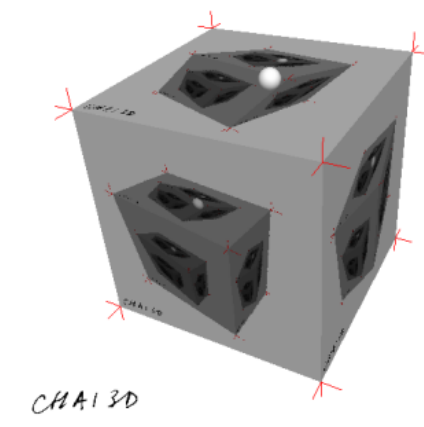
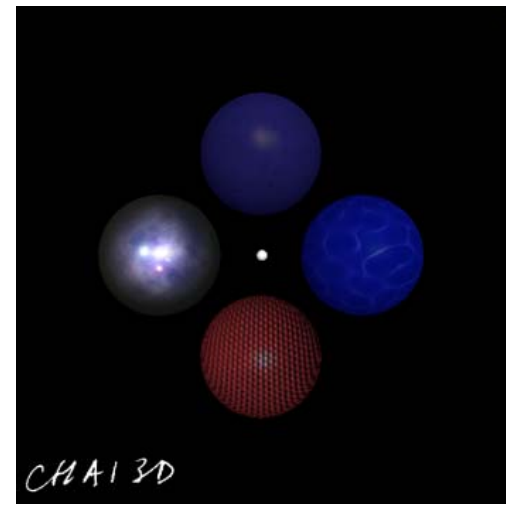
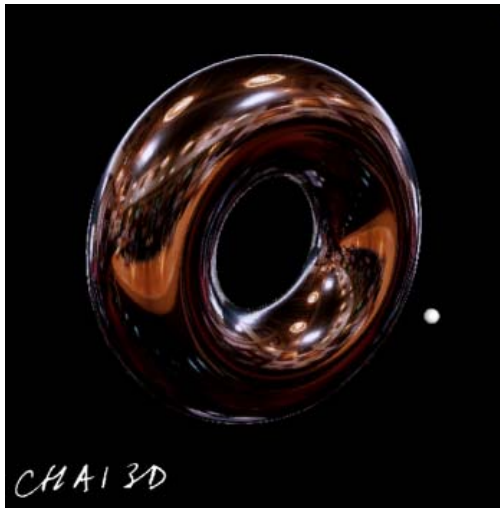
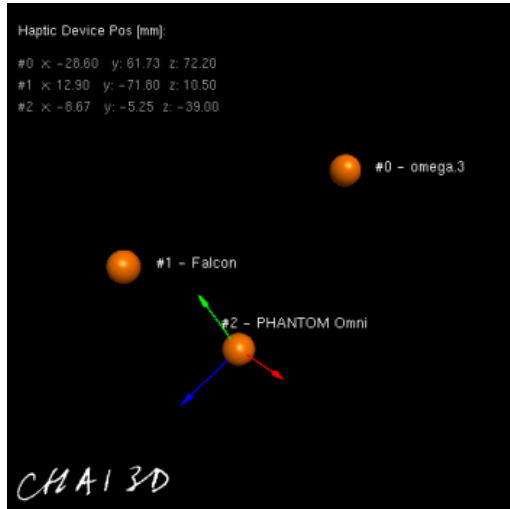
organization



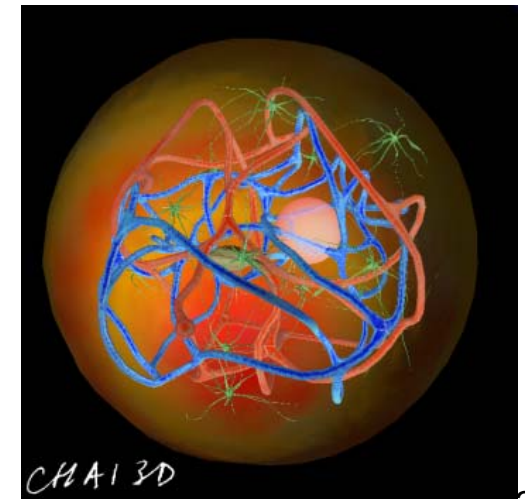
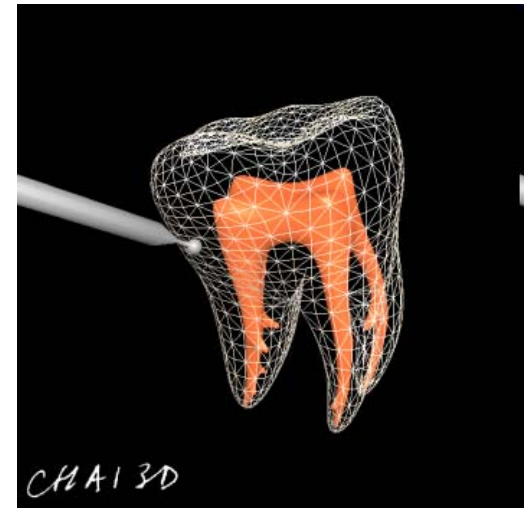
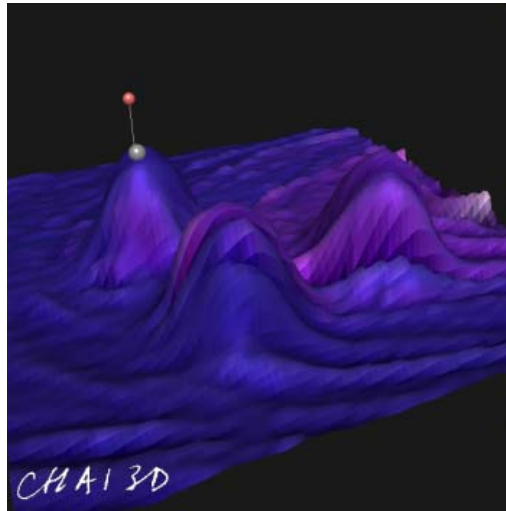
./examples



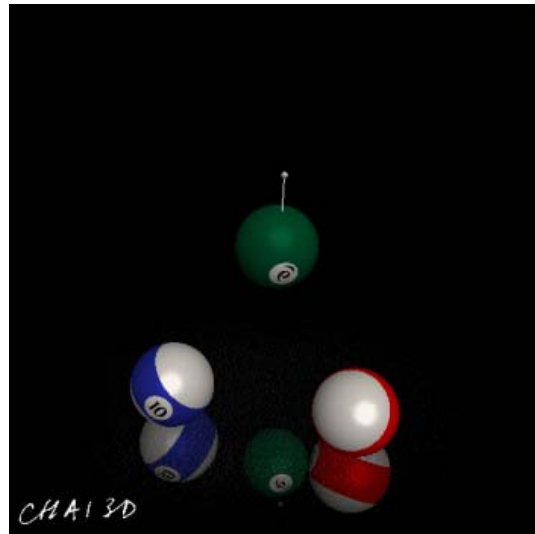
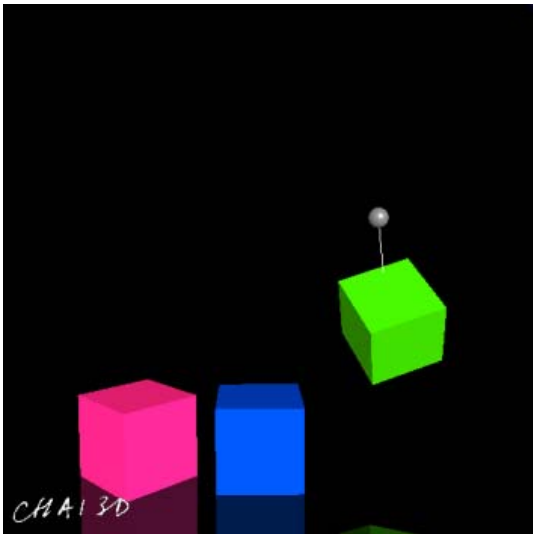
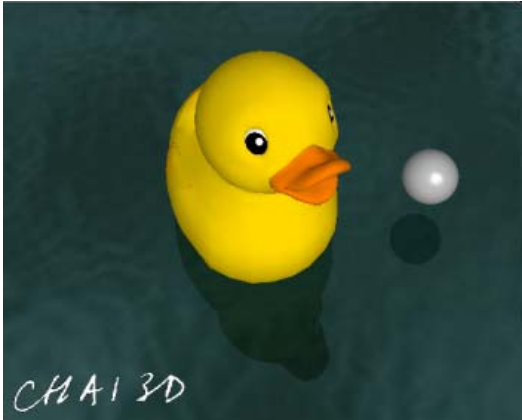
./examples



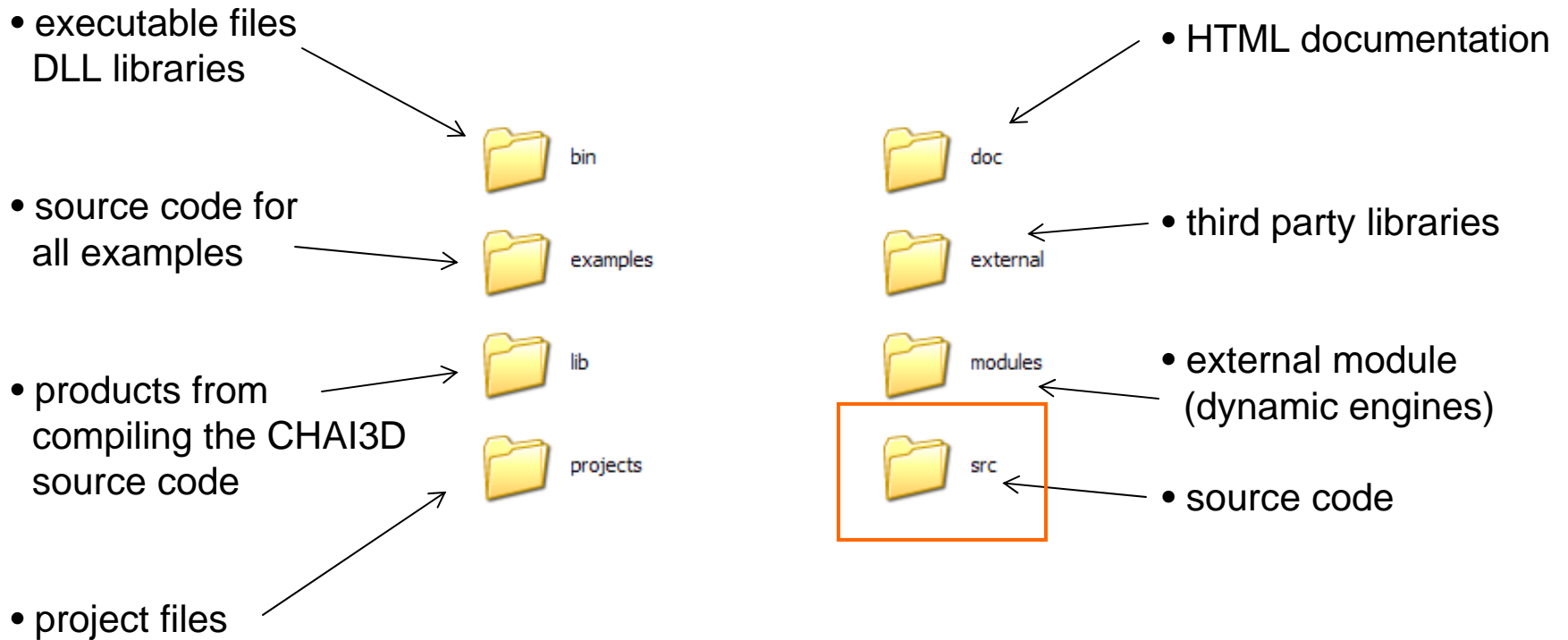
./examples



./examples

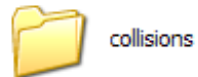


organization

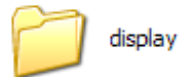


./src

- collision detection

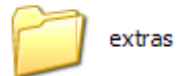


collisions



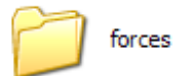
display

- configuration file



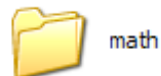
extras

- force rendering algorithms



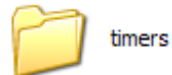
forces

- math, vectors, matrices



math

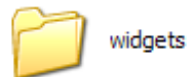
- threads & clocks



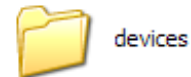
timers

- 2D objects

labels, fonts, images

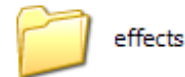


widgets



devices

- haptic devices



effects

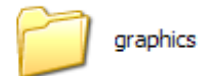
- haptic effects



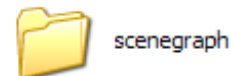
files

- file loaders

images, 3D models



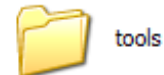
graphics



scenegraph

- virtual world

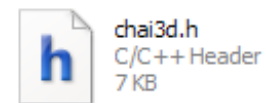
objects, tools, lights, etc...



tools

- virtual tool

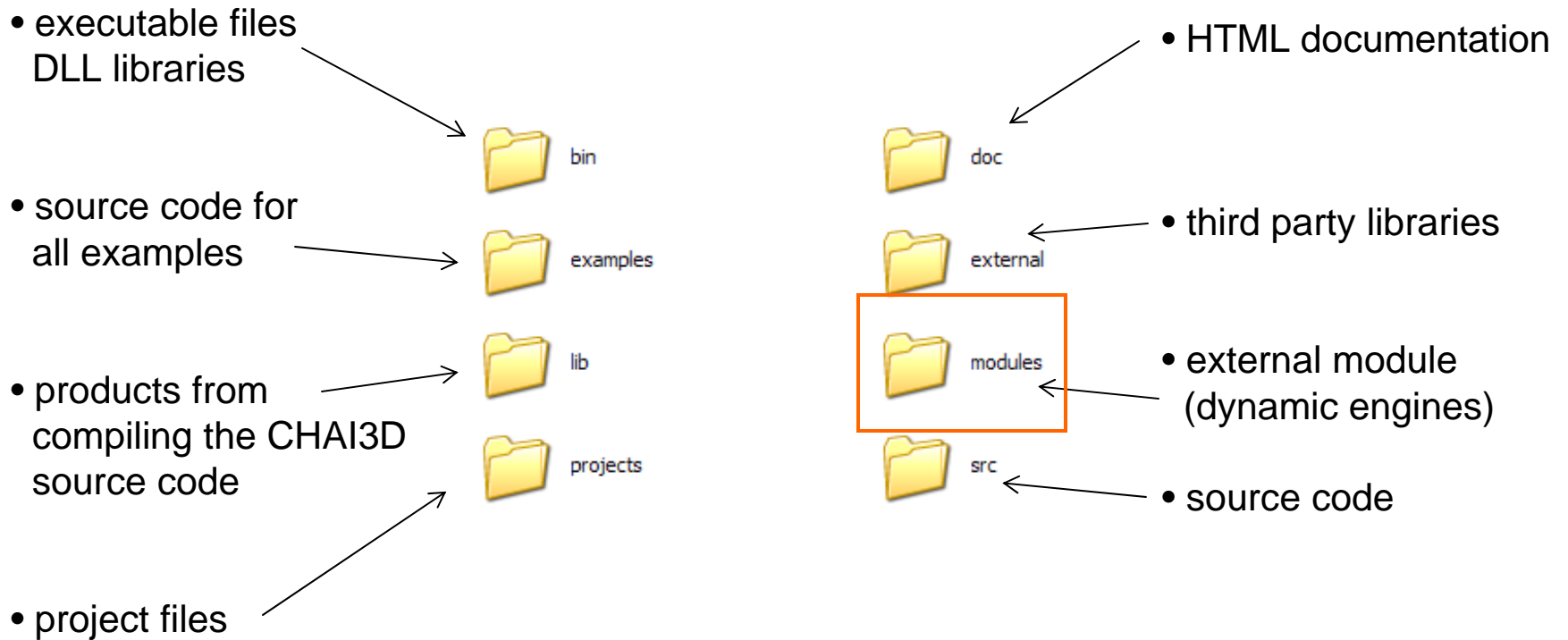
connect with your haptic device



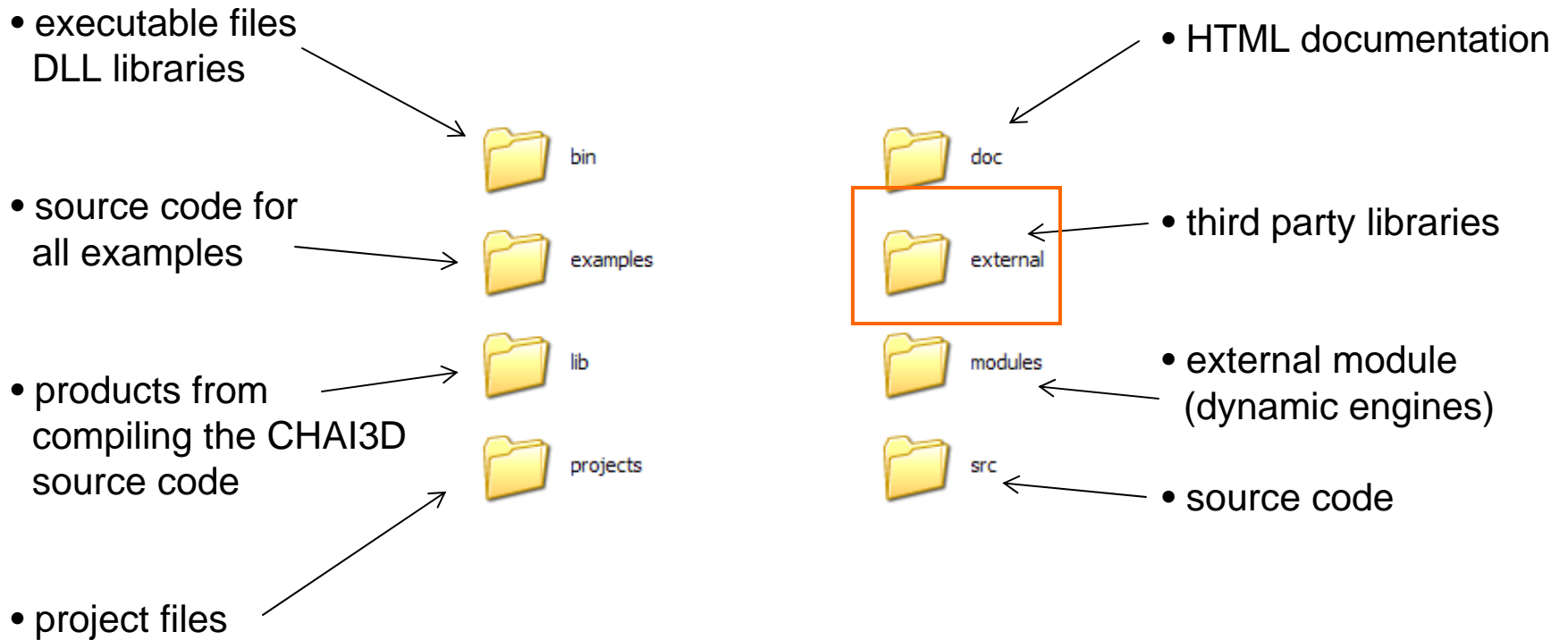
chai3d.h
C/C++ Header
7 KB

- main header file

organization



organization



outline

- organization of CHAI3D
- haptic device software interface
- creating a virtual world
- scene graph
- building object primitives
- programming haptic effects
- example
- developing your own primitives

haptic device interface

```
class cGenericHapticDevice
```

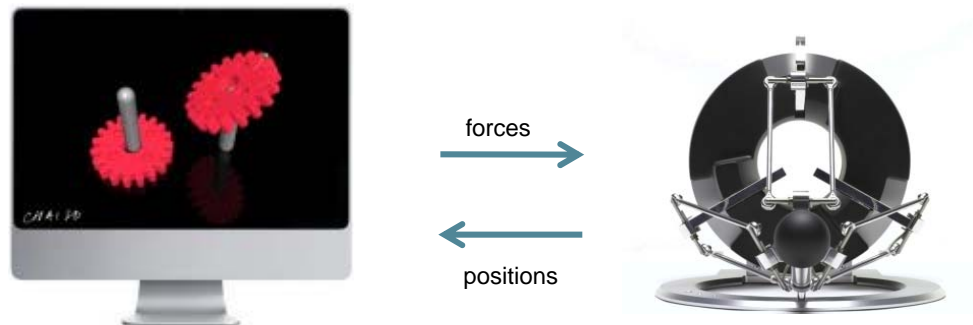
```
int open()  
int close()
```

```
int initialize()
```

```
int getPosition(cVector3d& a_position)  
int getLinearVelocity(cVector3d& a_linearVelocity)  
int getRotation(cMatrix3d& a_rotation)
```

```
int setForce(cVector3d& a_force)  
int getUserSwitch(int a_switchIndex, bool& a_status)
```

```
cHapticDeviceInfo getSpecifications()
```

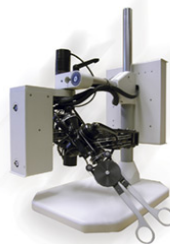


haptic device handler

application



haptic device handler (cHapticDeviceHandler)



connecting to a haptic device

```
// create a haptic device handler
handler = new cHapticDeviceHandler();

// get access to the first available haptic device
cGenericHapticDevice* hapticDevice;
handler->getDevice(hapticDevice, 0);

// retrieve information about the current haptic device
cHapticDeviceInfo info;
if (hapticDevice)
{
    info = hapticDevice->getSpecifications();
}

(...)
```

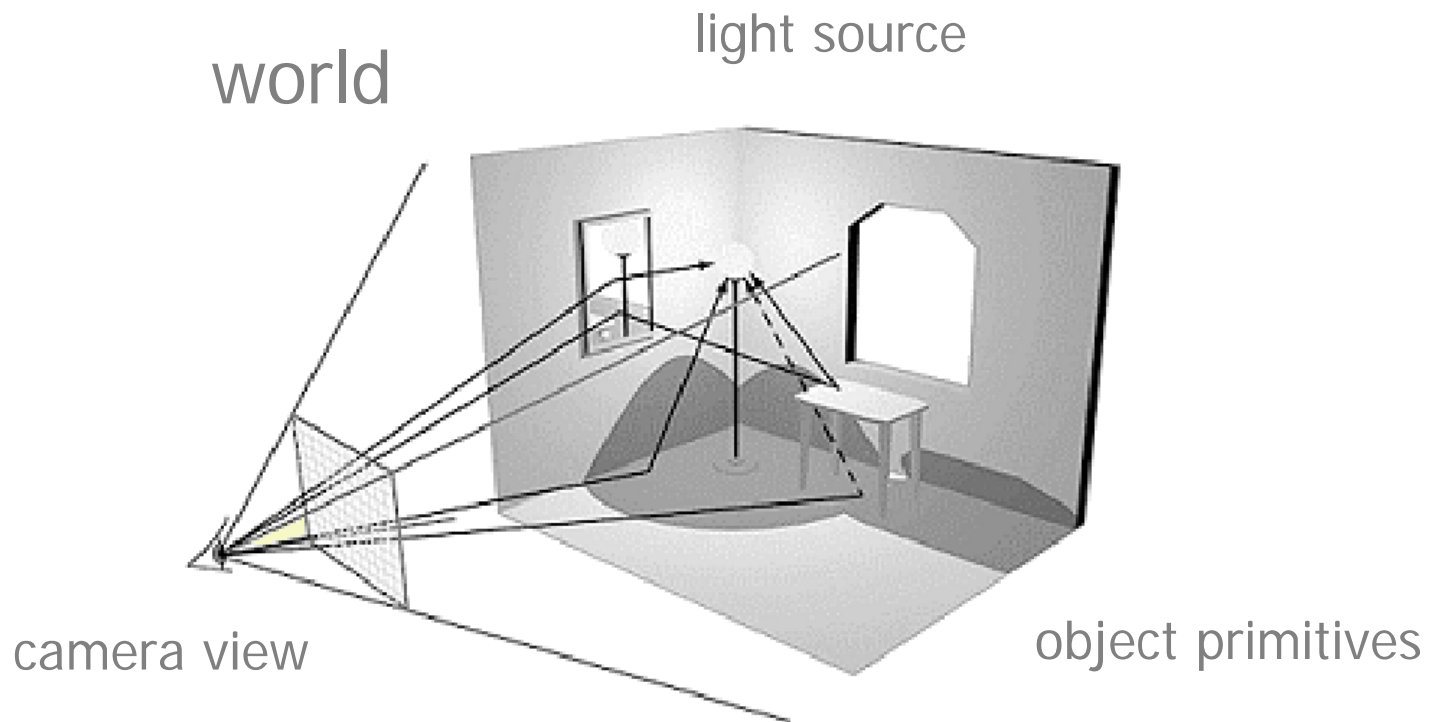
cHapticDeviceInfo

```
string m_manufacturerName;
double m_maxForce;
double m_maxForceStiffness;
double m_workspaceRadius;
bool m_sensedPosition;
bool m_sensedRotation;
bool m_actuatedPosition;
bool m_actuatedRotation;
(...)
```

outline

- organization of CHAI3D
- haptic device software interface
- creating a virtual world
- scene graph
- building object primitives
- programming haptic effects
- example
- developing your own primitives

virtual world



objects, materials and textures



mesh object (cMesh)



texture map (cTexture2D)



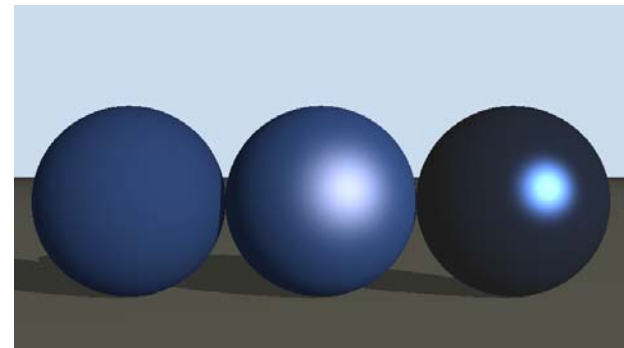
mesh object with
texture properties defined



mesh object with
wireframe display mode
activated



single textured triangle
and its 3 vertices



material properties
ambient, diffuse, specular components

material properties (cMaterial)

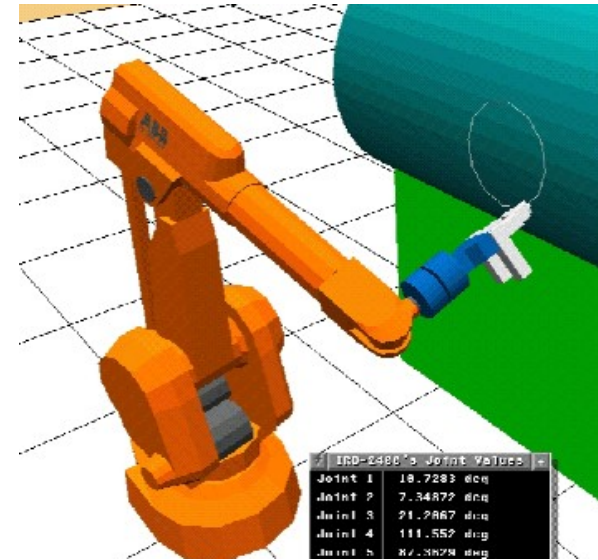
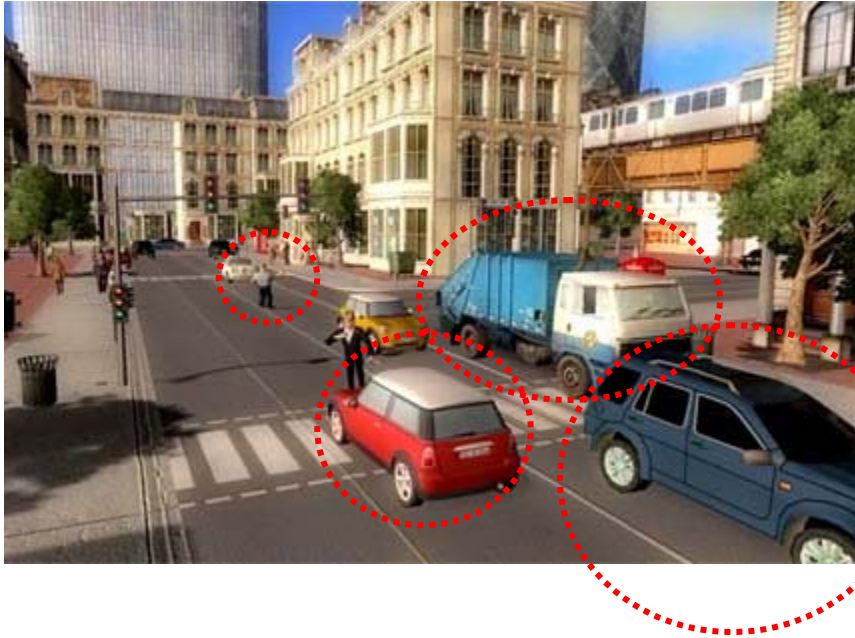
GRAPHIC PROPERTIES:

<code>cColorf m_ambient;</code>	Ambient color.
<code>cColorf m_diffuse;</code>	Diffuse color.
<code>cColorf m_specular;</code>	Specular color.
<code>cColorf m_emission;</code>	Emissive color.
<code>GLuint m_shininess;</code>	Shininess

HAPTIC PROPERTIES:

<code>double m_viscosity;</code>	Viscosity constant.
<code>double m_stiffness;</code>	Stiffness constant.
<code>double m_static_friction;</code>	Static friction constant.
<code>double m_dynamic_friction;</code>	Dynamic friction constant.
<code>double m_vibrationFrequency;</code>	Frequency of vibrations
<code>double m_vibrationAmplitude;</code>	Amplitude of vibrations.
<code>double m_magnetMaxForce;</code>	Maximum force applied by magnet effect.
<code>double m_stickSlipForceMax;</code>	Force threshold for stick and slip effect.
<code>double m_stickSlipStiffness;</code>	Spring stiffness of stick slip model.

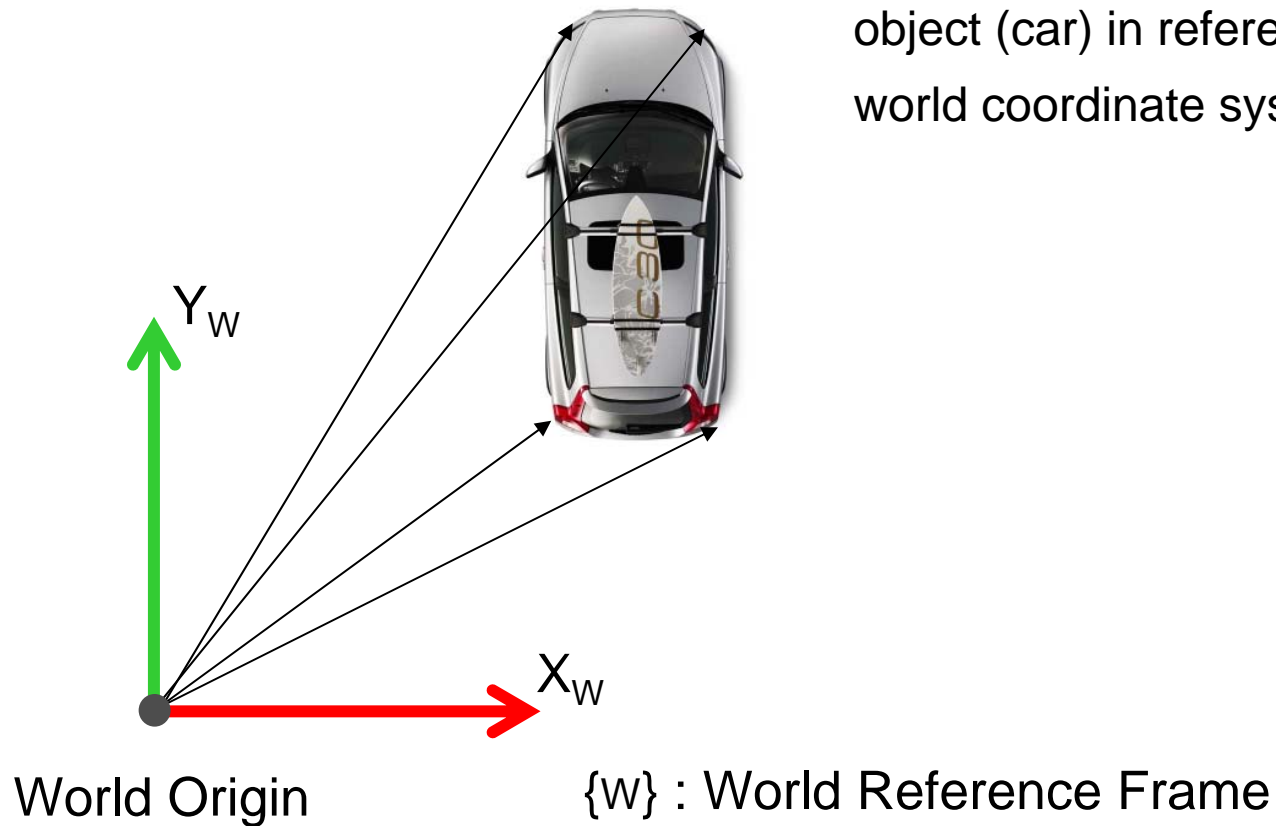
reference frames



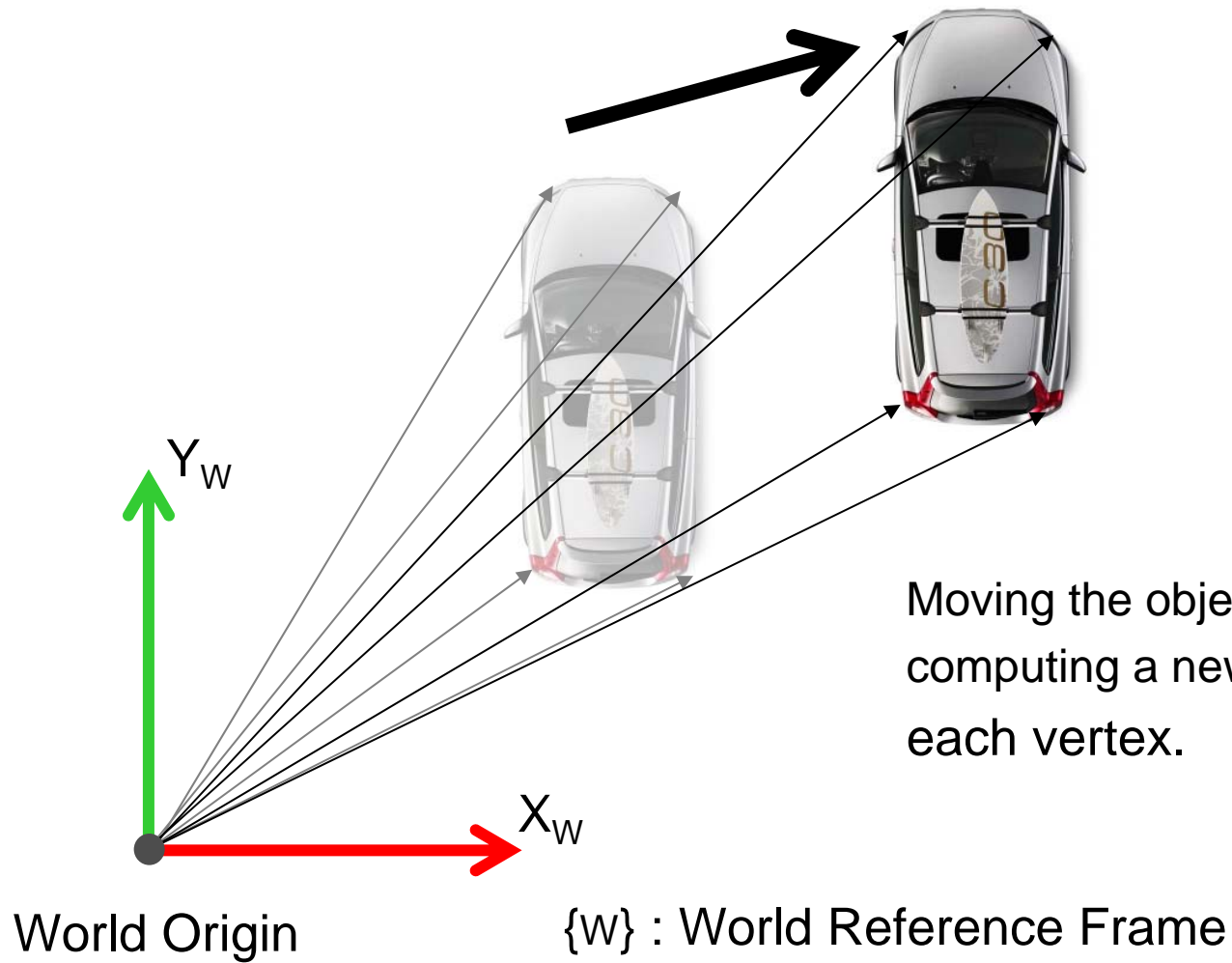
- Defining independent objects in the world
- Defining relationships between these objects

reference frames

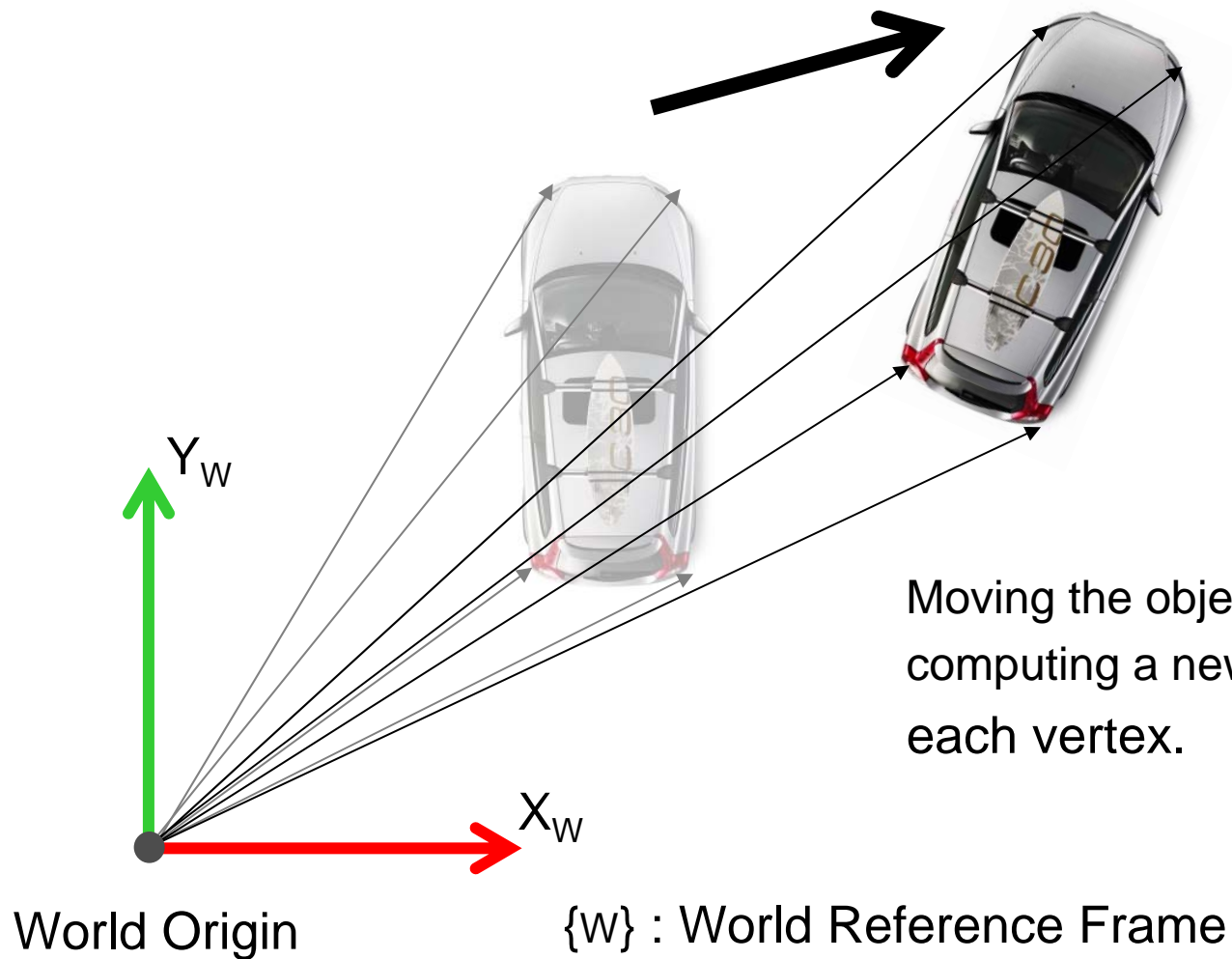
Expressing the vertices of the object (car) in reference with the world coordinate system.



reference frames

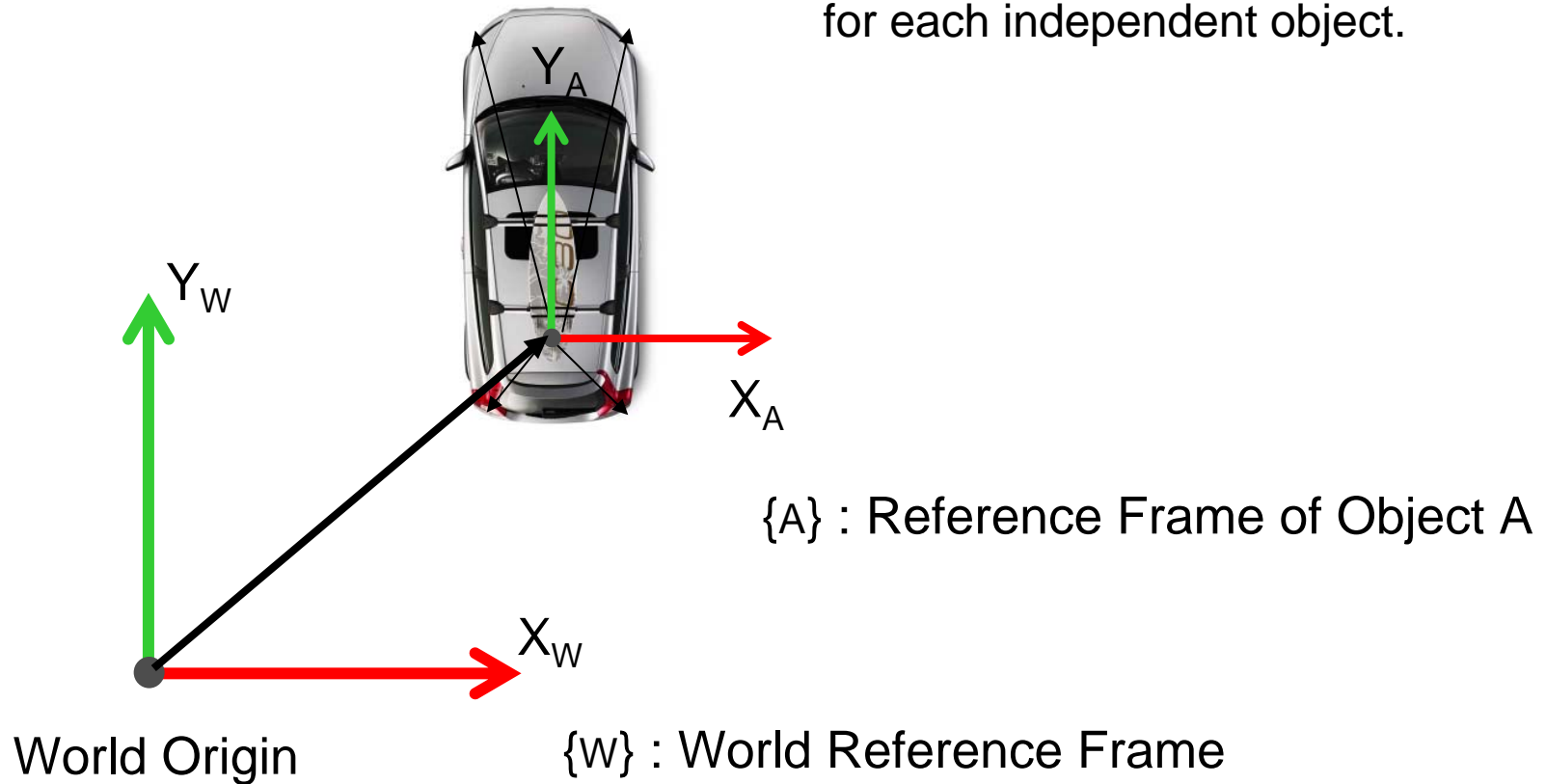


reference frames

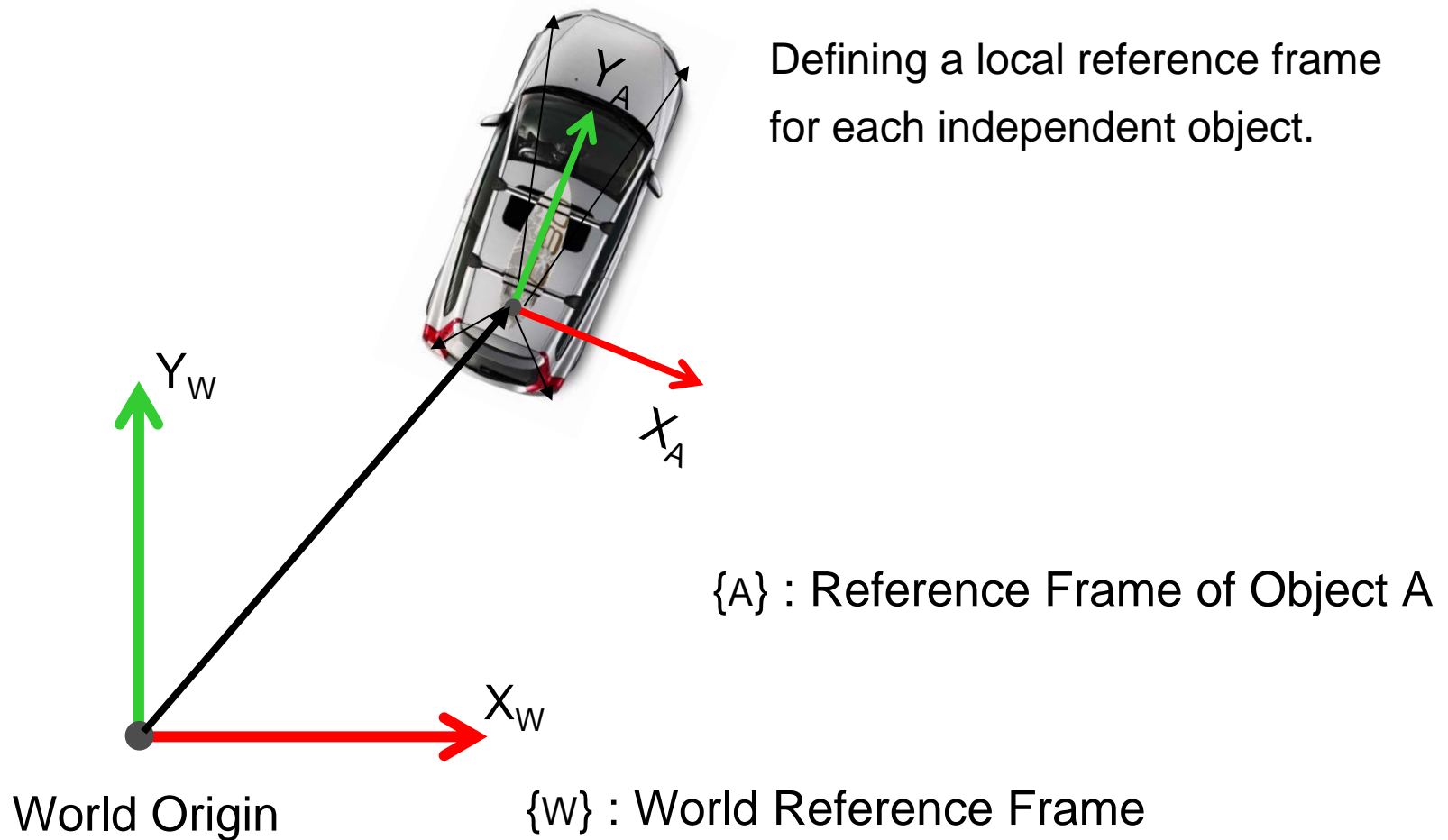


reference frames

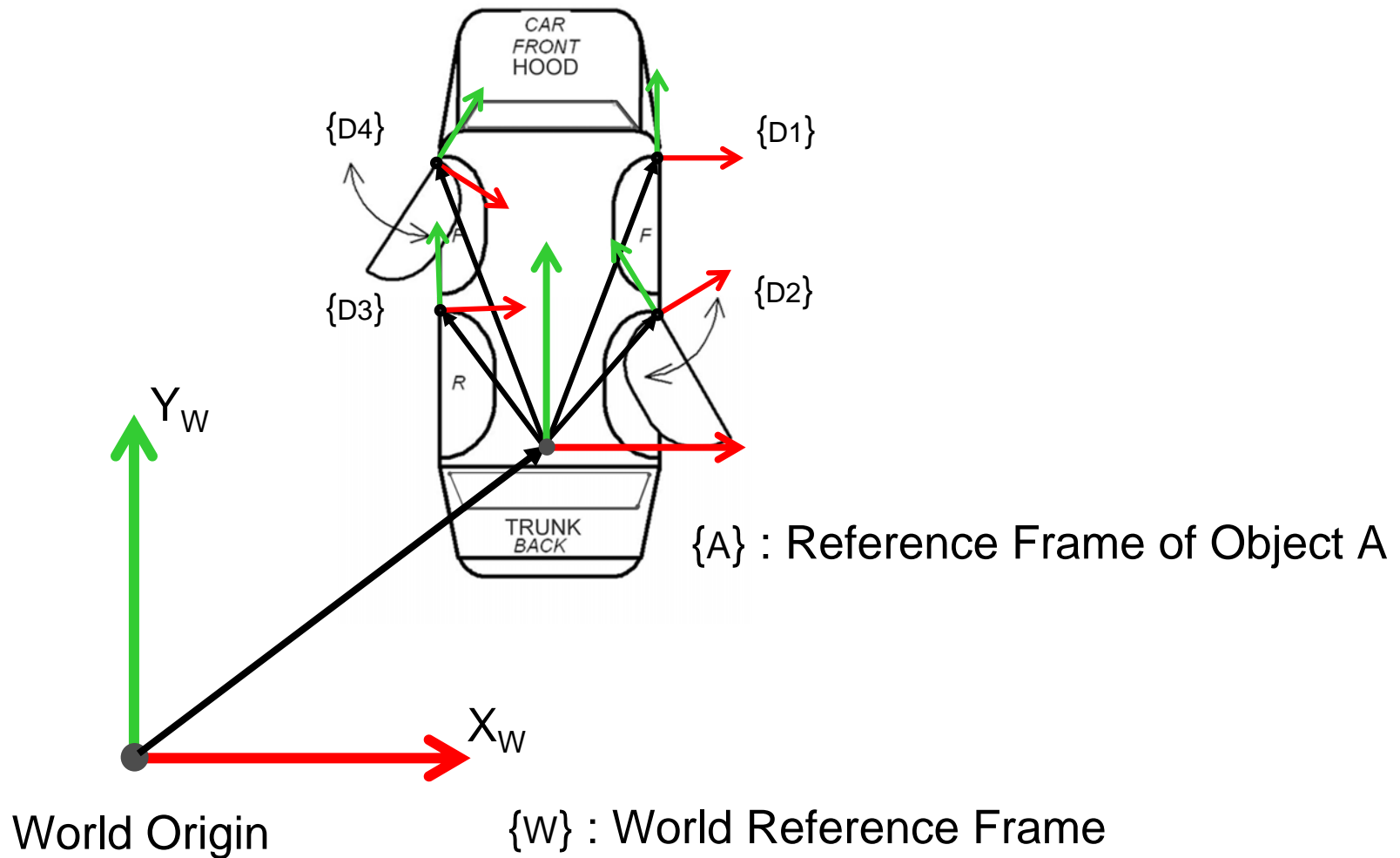
Defining a local reference frame for each independent object.



reference frames

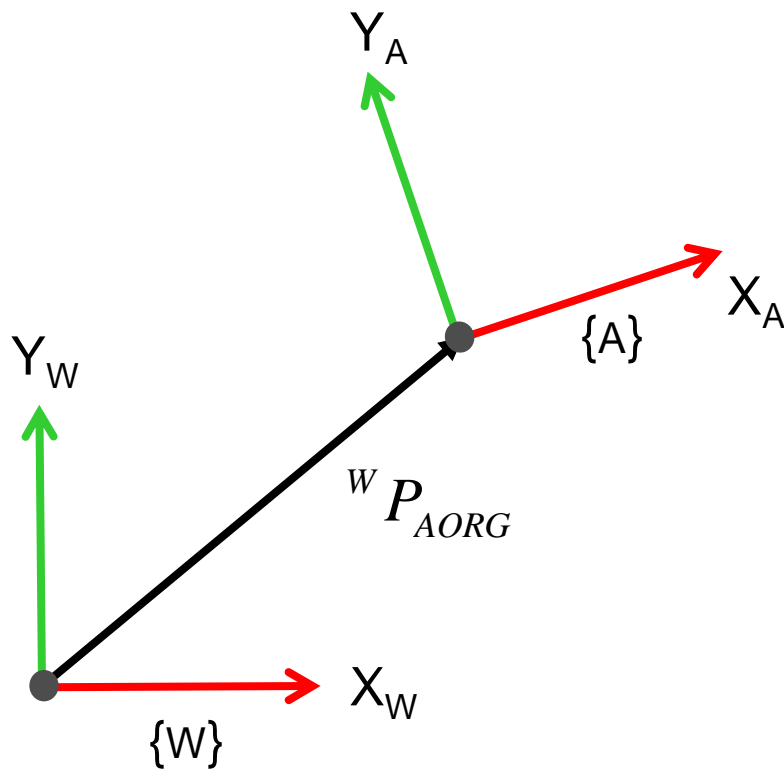


reference frames



reference frames

Translation: $\{W\} \rightarrow \{A\}$

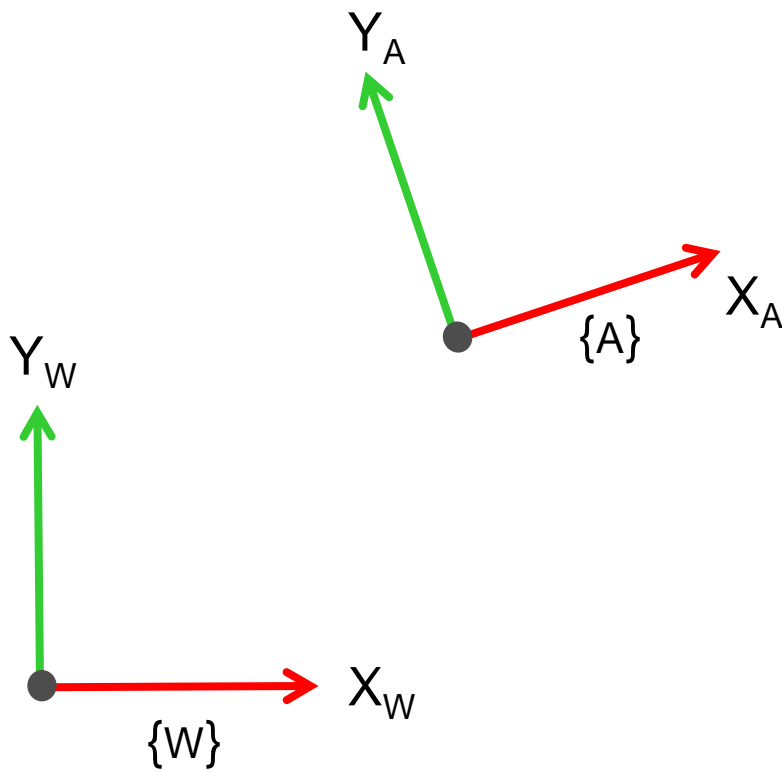


Vector:

$${}^W P_{AORG} = \begin{bmatrix} {}^W P_x \\ {}^W P_y \end{bmatrix}$$

reference frames

Rotation: $\{W\} \rightarrow \{A\}$

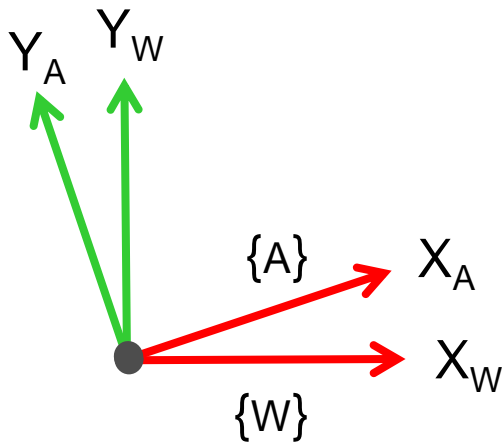


reference frames

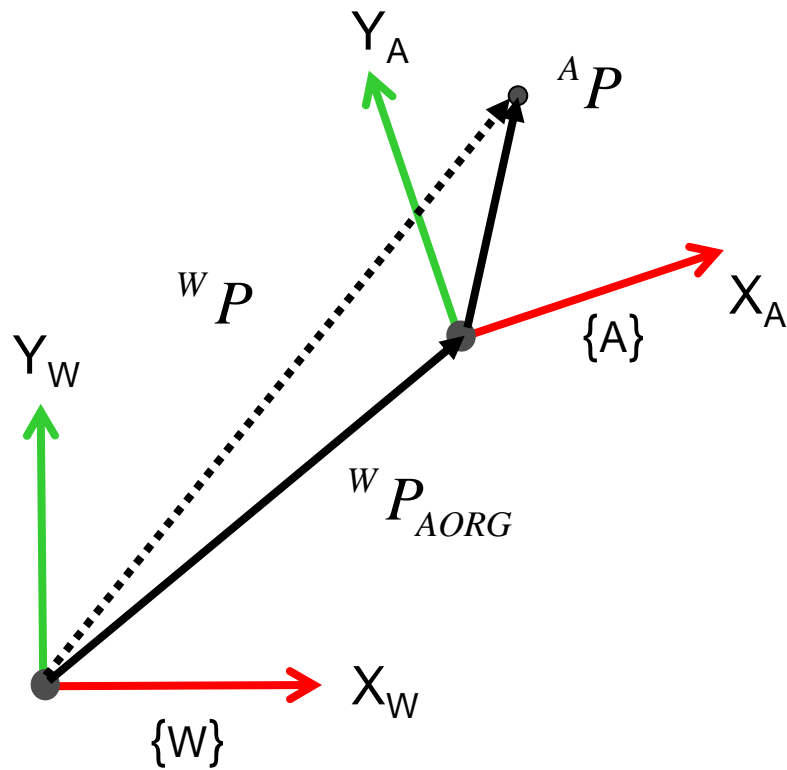
Rotation: $\{W\} \rightarrow \{A\}$

Rotation Matrix:

$${}^W_A R = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} = \begin{bmatrix} {}^W X_A & {}^W Y_A \end{bmatrix}$$

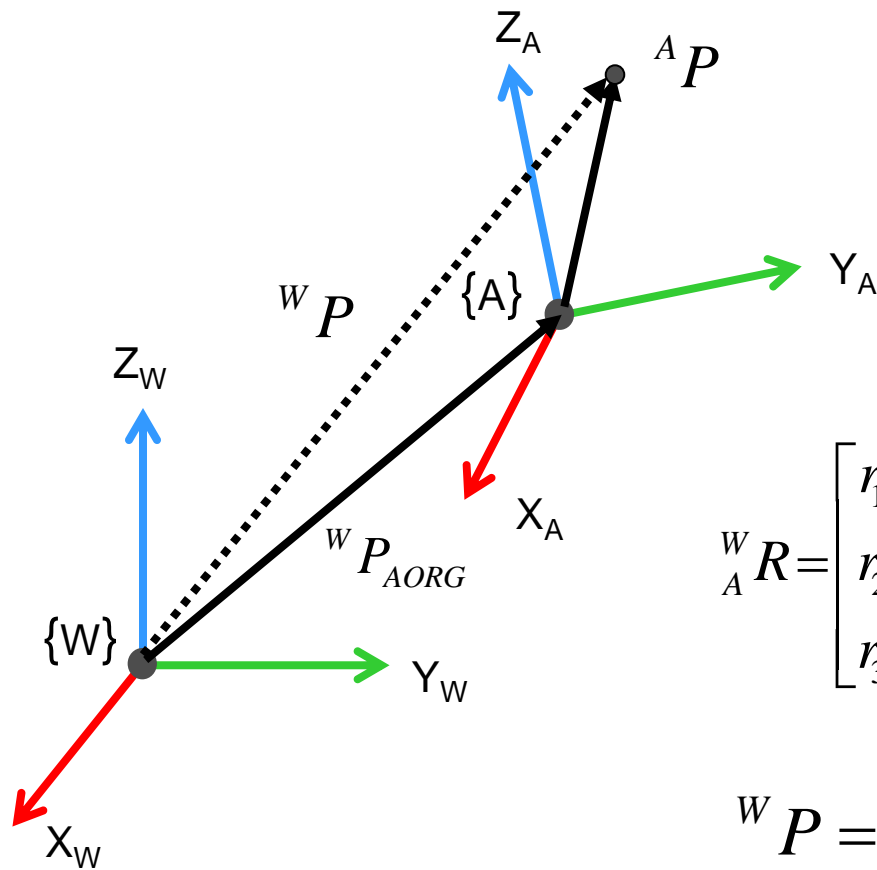


reference frames



$${}^W P = {}^W_A R {}^A P + {}^W P_{AORG}$$

reference frames



$${}^W_A R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} {}^W X_A & {}^W Y_A & {}^W Z_A \end{bmatrix}$$

$${}^W P = {}^W_A R {}^A P + {}^W P_{AORG}$$

reference frames

$${}^W P = {}^W_A R {}^A P + {}^W P_{AORG}$$

Homogeneous Transform:

$$\begin{bmatrix} {}^W P \\ \hline 1 \end{bmatrix} = \begin{bmatrix} {}^W_A R & {}^W P_{AORG} \\ \hline 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^A P \\ \hline 1 \end{bmatrix}$$

- A “1” is added as the last element of the 4x1 vector.
- A row of “[0 0 0 1]” is added as the last row of the 4x4 matrix

$${}^W P = {}^W_A T \cdot {}^A P$$

 **Transform:** Translation and Rotation

reference frames

 ${}^W_A T$

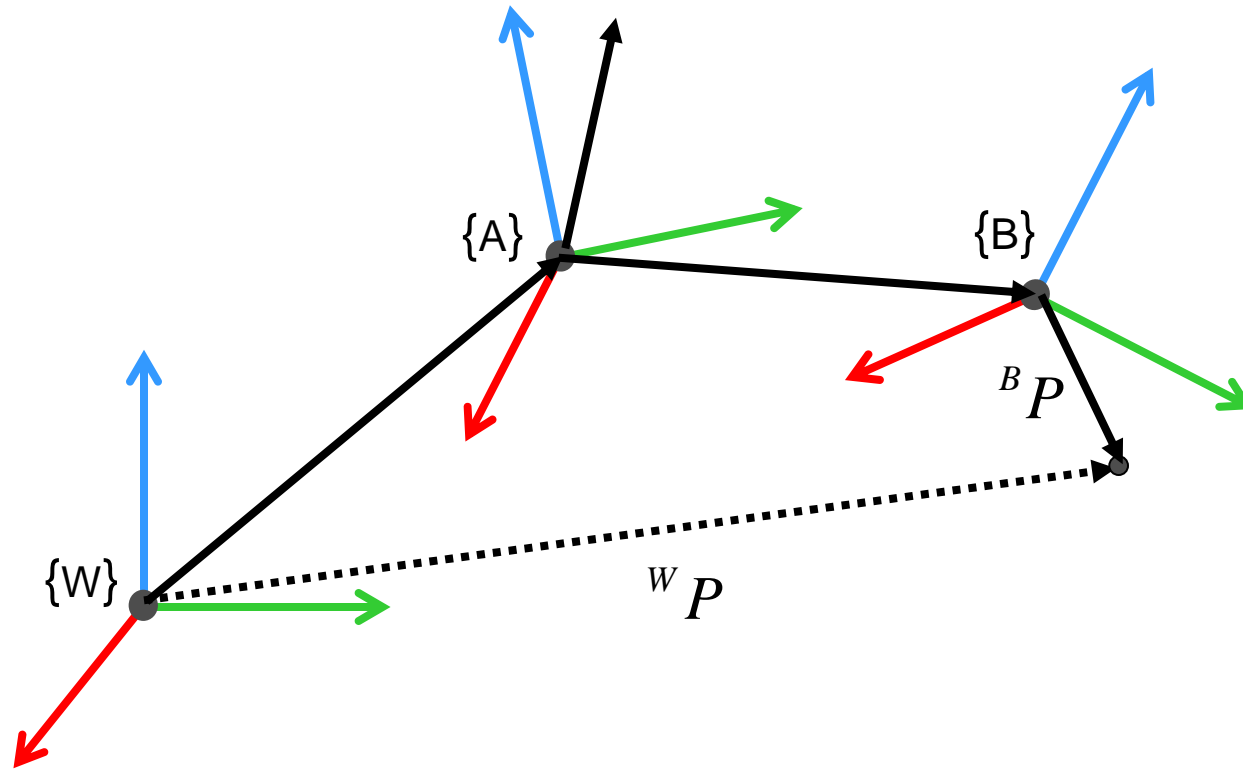
$$\left[\begin{array}{ccc|c} & & & \\ & {}^W_A R & & {}^W P_{AORG} \\ & & & \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

Inverting a Transform:

 ${}^A_W T$

$$\left[\begin{array}{ccc|c} & & & \\ & {}^W_A R^T & & -{}^W_A R^T \cdot {}^W P_{AORG} \\ & & & \\ \hline 0 & 0 & 0 & 1 \end{array} \right]$$

reference frames



$${}^W P = {}^W T_A {}^A T_B {}^B P$$

reference frames

With homogenous coordinates all affine transformations reduces to a single matrix multiplication.

Computational efficiency:

Standard coordinates

- Matrix-vector multiplication 3x3:
 - 9 multiplications + 6 additions
- Vector-vector addition 3x1:
 - 3 additions

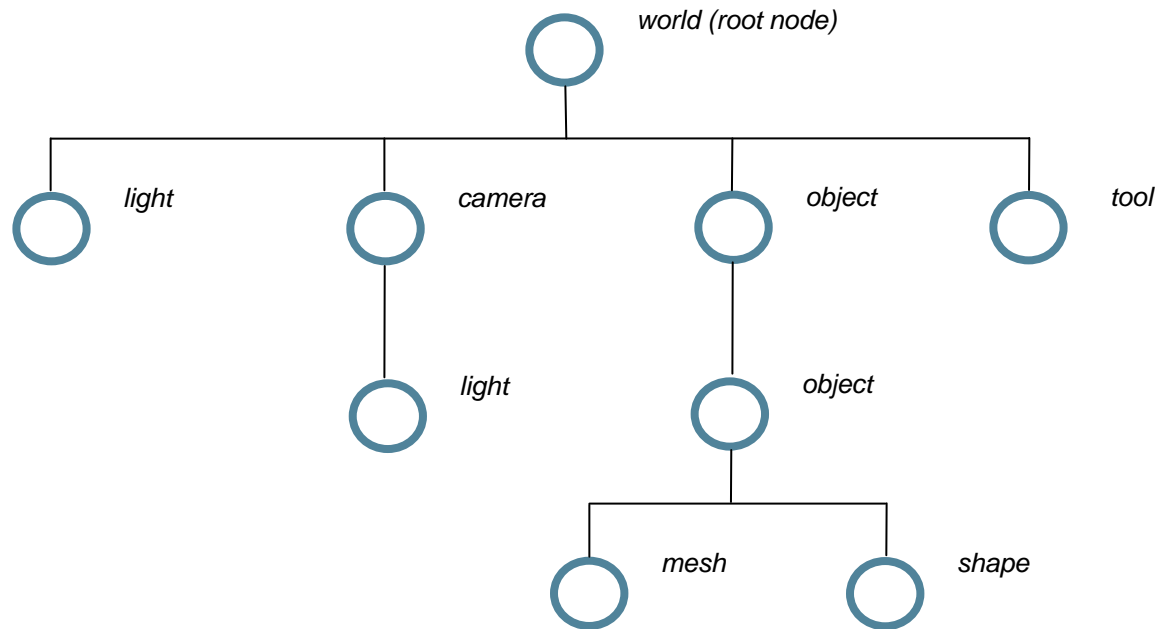
$${}^W P = {}^W_A R {}^A P + {}^W P_{AORG}$$

Homogenous coordinates

- Matrix-vector multiplication 4x4:
 - 16 multiplications + 12 additions

$${}^W P = {}^W_A T {}^A P$$

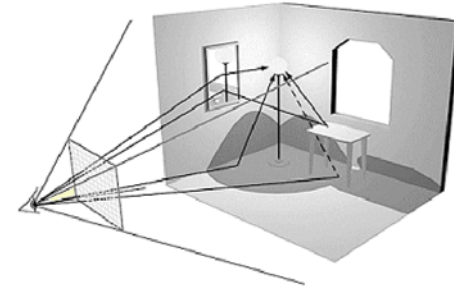
scene graph



fundamental nodes

basic scenegraph node

cGenericObject



world (root node)

cWorld

camera view

cCamera

light sources

cLight

object primitives

cShapeSphere

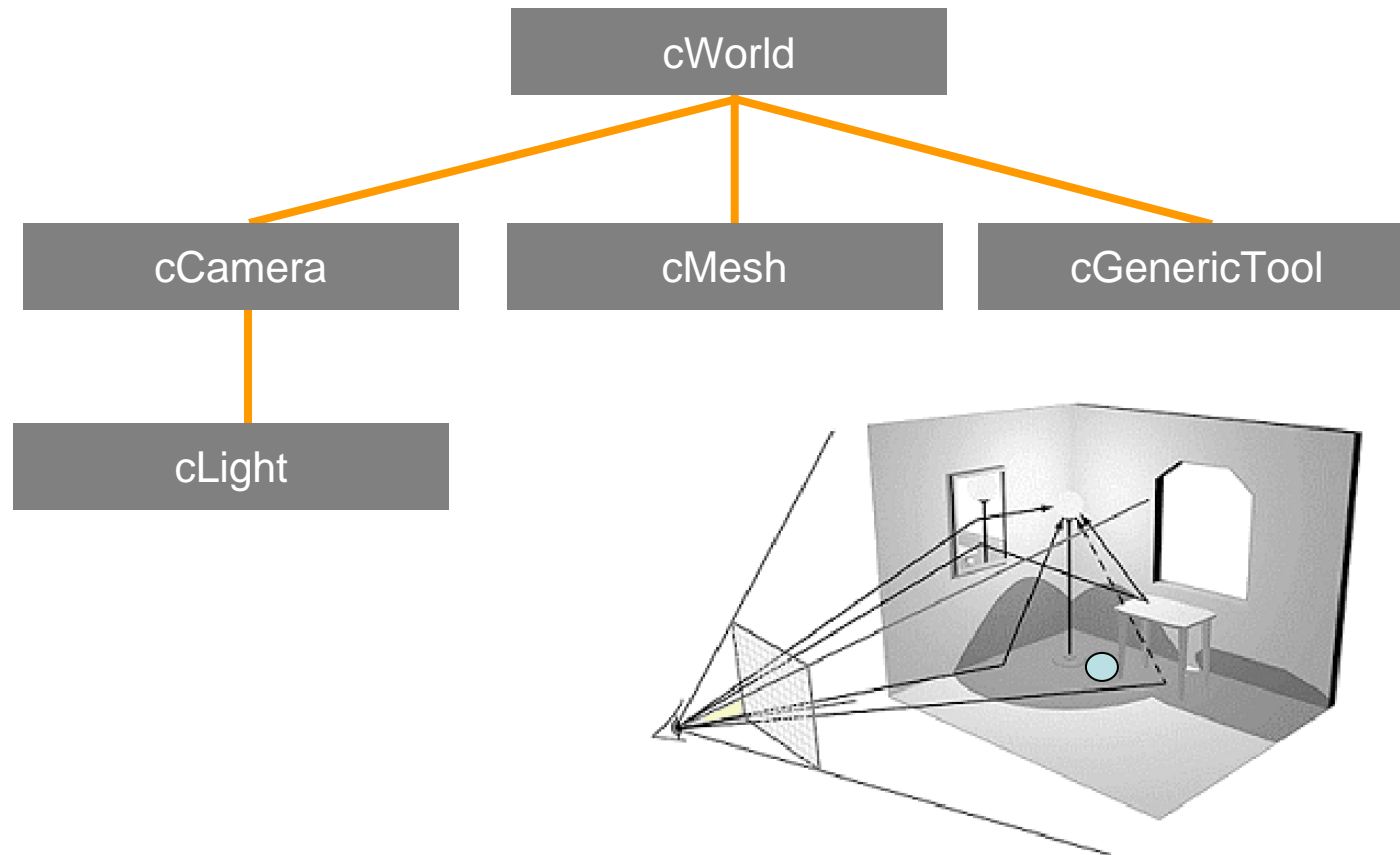
cShapeTorus

cMesh

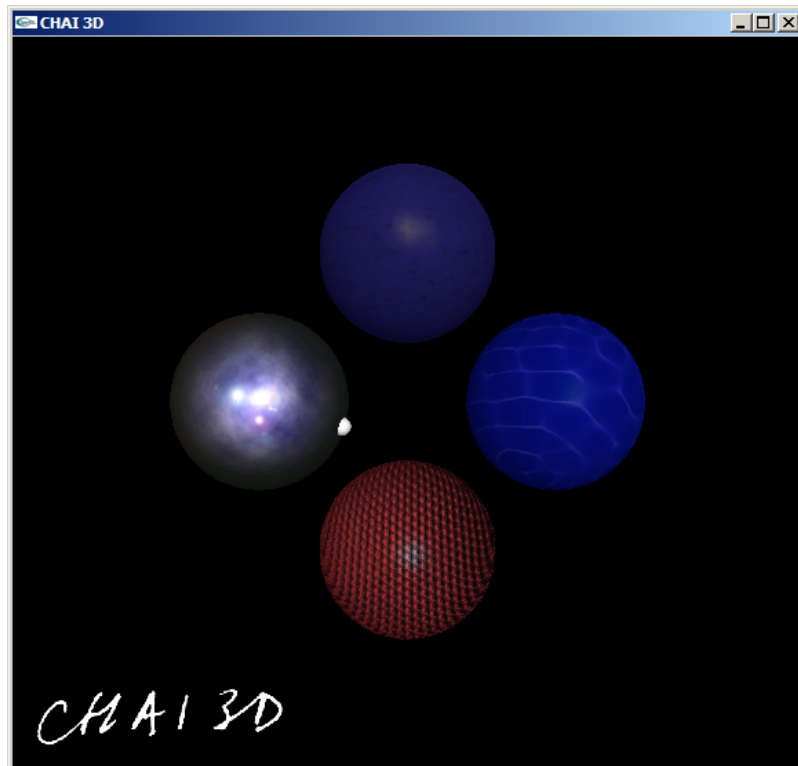
haptic tools

cGenericTool

scene graph - example



example



example

```
//=====
/*
    DEMO:    effects.cpp
    In this example illustrates the use of haptic effects. The application
    begins by creating four spheres with different graphical (
    material and texture) properties. For each sphere, one or more effects
    are programmed to obtain the desired haptic illusion. Parameters to
    adjust each effect are located in the cMaterial class.
    In the main haptics loop function "updateHaptics()" , the position
    of the haptic device is retrieved at each simulation iteration.
    The interaction forces are then computed and sent to the haptic device.
*/
//=====
//-----
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//-----
#include "chai3d.h"
//-----
```

example

```
//-----  
// DECLARED CONSTANTS  
//-----  
  
// initial size (width/height) in pixels of the display window  
const int WINDOW_SIZE_W      = 600;  
const int WINDOW_SIZE_H      = 600;  
  
// mouse menu options (right button)  
const int OPTION_FULLSCREEN   = 1;  
const int OPTION_WINDOWDISPLAY = 2;
```

example

```
//-----  
// DECLARED VARIABLES  
//-----  
// a world that contains all objects of the virtual environment  
cWorld* world;  
  
// a camera that renders the world in a window display  
cCamera* camera;  
  
// a light source to illuminate the objects in the virtual scene  
cLight *light;  
  
// width and height of the current window display  
int displayW = 0;  
int displayH = 0;  
  
// a haptic device handler  
cHapticDeviceHandler* handler;  
  
// a virtual tool representing the haptic device in the scene  
cGeneric3dofPointer* tool;  
  
// a few spherical objects  
cShapeSphere* object0;  
cShapeSphere* object1;  
cShapeSphere* object2;  
cShapeSphere* object3;  
  
// status of the main simulation haptics loop  
bool simulationRunning = false;  
  
// has exited haptics simulation thread  
bool simulationFinished = false;
```

example

```
//-----  
// DECLARED FUNCTIONS  
//-----  
  
// callback when the window display is resized  
void resizeWindow(int w, int h);  
  
// callback when a keyboard key is pressed  
void keySelect(unsigned char key, int x, int y);  
  
// callback when the right mouse button is pressed to select a menu item  
void menuSelect(int value);  
  
// function called before exiting the application  
void close(void);  
  
// main graphics callback  
void updateGraphics(void);  
  
// main haptics loop  
void updateHaptics(void);
```

example

```
//-----  
// MAIN APPLICATION  
//-----  
  
int main(int argc, char* argv[])  
{  
  
    //-----  
    // INITIALIZATION  
    //-----  
    printf ("\n");  
    printf ("-----\n");  
    printf ("CHAI 3D\n");  
    printf ("Demo: 11-effects\n");  
    printf ("Copyright 2003-2010\n");  
    printf ("-----\n");  
  
    //-----  
    // 3D - SCENEGRAPH  
    //-----  
  
    // create a new world.  
    world = new cWorld();  
  
    // set the background color of the environment  
    // the color is defined by its (R,G,B) components.  
    world->setBackgroundColor(0.0, 0.0, 0.0);  
}
```

Here we setup the basic primitives of our scene, namely:

- **world** – the main root of our scenegraph
- **camera** – to view the world
- **light** source – to see what is going on in the scene!

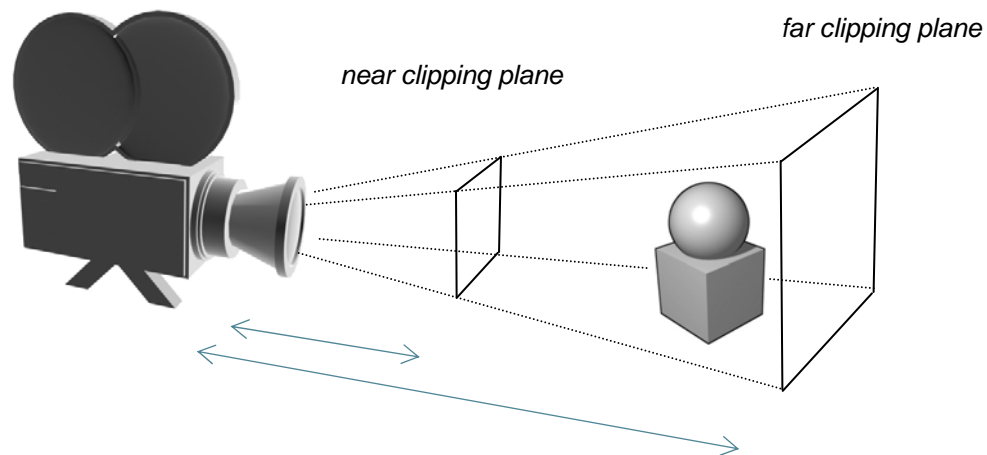
example

```
// create a camera and ass it to the world
camera = new cCamera(world);
world->addChild(camera);

// position and oriente the camera
camera->set( cVector3d (3.0, 0.0, 0.0), // camera position (eye)
           cVector3d (0.0, 0.0, 0.0), // lookat position (target)
           cVector3d (0.0, 0.0, 1.0)); // direction of the "up" vector

// set the near and far clipping planes of the camera
// anything in front/behind these clipping planes will not be rendered
camera->setClippingPlanes(0.01, 10.0);

// enable higher quality rendering for transparent objects
camera->enableMultipassTransparency(true);
```



example

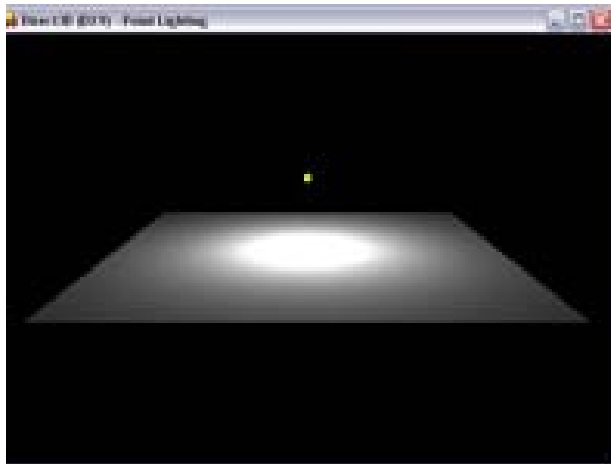
```
// create a light source and attach it to the camera
light = new cLight(world);

// attach light to camera
camera->addChild(light);

// enable light source
light->setEnabled(true);

// position the light source
light->setPos(cVector3d( 2.0, 0.5, 1.0));

// define the direction of the light beam
light->setDir(cVector3d(-2.0, 0.5, 1.0));
```



example

```
//-----  
// HAPTIC DEVICE  
//-----  
  
// create a haptic device handler  
handler = new cHapticDeviceHandler();  
  
// get access to the first available haptic device  
cGenericHapticDevice* hapticDevice;  
handler->getDevice(hapticDevice, 0);  
  
// retrieve information about the current haptic device  
cHapticDeviceInfo info;  
info = hapticDevice->getSpecifications();
```

Here we search for a haptic device that is connected to the computer and retrieve information about its characteristics



example

```
// create a 3D tool and add it to the world
tool = new cGeneric3dofPointer(world);
world->addChild(tool);

// connect the haptic device to the tool
tool->setHapticDevice(hapticDevice);

// initialize tool by connecting to haptic device
tool->start();

// map the physical workspace of the haptic device to a larger
// virtual workspace.
tool->setWorkspaceRadius(1.0);

// define a display radius of the cursor
tool->setRadius(0.03);

// read the scale factor between the physical workspace of the haptic
// device and the virtual workspace defined for the tool
double workspaceScaleFactor = tool->getWorkspaceScaleFactor();

// define a maximum stiffness that can be handled by the current
// haptic device. The value is scaled to take into account the
// workspace scale factor
double stiffnessMax = info.m_maxForceStiffness / workspaceScaleFactor;
double forceMax = info.m_maxForce;

// define the maximum damping factor that can be handled by the
// current haptic device. The The value is scaled to take into account the
// workspace scale factor
double dampingMax = info.m_maxLinearDamping / workspaceScaleFactor;
```

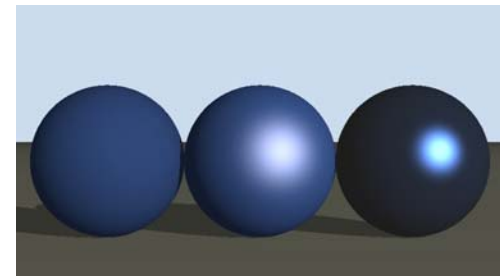
Some haptic devices are smaller than others so we need to map their physical workspace to the workspace of the virtual environment. At the same time we also adjust the stiffness properties of the objects in the scene.

example

```
//-----  
// CREATE A VIRTUAL OBJECT  
//-----  
  
// create a sphere and define its radius  
object0 = new cShapeSphere(0.3);  
  
// add object to world  
world->addChild(object0);  
  
// set the position of the object at the center of the world  
object0->setPos(0.0, -0.5, 0.0);  
  
// create some texture properties  
object0->m_texture = new cTexture2D();  
object0->m_texture->loadFromFile("resources/images/chrome.bmp");  
  
// enable spherical mapping (see texture properties)  
object0->m_texture->setSphericalMappingEnabled(true);  
  
// define some material properties  
object0->m_material.m_ambient.set(1.0, 1.0, 1.0);  
object0->m_material.m_diffuse.set(1.0, 1.0, 1.0);  
object0->m_material.m_specular.set(1.0, 1.0, 1.0);  
object0->m_material.setShininess(100);  
object0->setUseTexture(true);  
  
// define some haptic properties  
object0->m_material.setStiffness(0.4 * stiffnessMax);  
object0->m_material.setMagnetMaxForce(0.4 * forceMax);  
object0->m_material.setMagnetMaxDistance(0.12);  
object0->m_material.setViscosity(0.2 * dampingMax);  
  
// create a haptic surface effect  
newEffect = new cEffectSurface(object0);  
object0->addEffect(newEffect);  
  
// create a haptic magnetic effect  
newEffect = new cEffectMagnet(object0);  
object0->addEffect(newEffect);
```

Here we create a virtual sphere by defining its size and its material properties. We also cover it with texture to it by loading a texture map file.

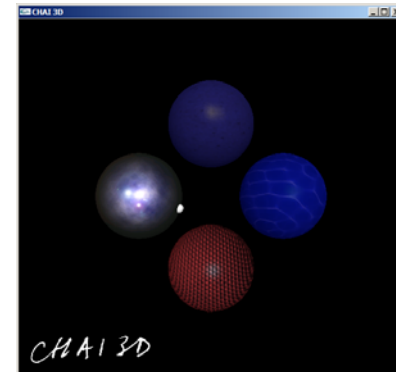
Finally we also program two haptic effect to make it behaves like a magnet.



example

```
//-----  
// OPEN GL - WINDOW DISPLAY  
//-----  
  
// initialize GLUT  
glutInit(&argc, argv);  
  
// defining position of window within computer screen  
int screenW = glutGet(GLUT_SCREEN_WIDTH);  
int screenH = glutGet(GLUT_SCREEN_HEIGHT);  
int windowPosX = (screenW - WINDOW_SIZE_W) / 2;  
int windowPosY = (screenH - WINDOW_SIZE_H) / 2;  
  
// initialize the OpenGL GLUT window  
glutInitWindowPosition(windowPosX, windowPosY);  
glutInitWindowSize(WINDOW_SIZE_W, WINDOW_SIZE_H);  
glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);  
glutCreateWindow(argv[0]);  
glutDisplayFunc(updateGraphics);  
glutKeyboardFunc(keySelect);  
glutReshapeFunc(resizeWindow);  
  
// create a mouse menu (right button)  
glutCreateMenu(menuSelect);  
glutAddMenuEntry("full screen", OPTION_FULLSCREEN);  
glutAddMenuEntry("window display", OPTION_WINDOWDISPLAY);  
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

This code here create the GLUT main window and attaches the various callbacks to render graphics, handle window resizing, mouse clicks, and keyboard commands.



example

```
//-----  
// START SIMULATION  
//-----  
  
// simulation is now running  
simulationRunning = true;  
  
// create a thread which starts the main haptics rendering loop  
cThread* hapticsThread = new cThread();  
hapticsThread->set(updateHaptics, CHAI_THREAD_PRIORITY_HAPTICS);  
  
// start the main graphics rendering loop  
glutMainLoop();  
  
// close everything  
close();  
  
// exit  
return (0);  
}
```

We are finally ready and we can now start our haptics thread!

example

```
void resizeWindow(int w, int h)
{
    // update the size of the viewport
    displayW = w;
    displayH = h;
    glViewport(0, 0, displayW, displayH);
}
```

This GLUT window callback informs the application the user has just resized the main Window display. We now inform OpenGL about the new dimensions of the display (viewport)

```
//-----
```

```
void keySelect(unsigned char key, int x, int y)
{
    if ((key == 27) || (key == 'x'))
    {
        // close everything
        close();

        // exit application
        exit(0);
    }
}
```

This GLUT window callback handles key commands coming from the keyboard.

Here we terminate the application when the user presses the Escape key.

example

```
void menuSelect(int value)
{
    switch (value)
    {
        // enable full screen display
        case OPTION_FULLSCREEN:
            glutFullScreen();
            break;

        // reshape window to original size
        case OPTION_WINDOWDISPLAY:
            glutReshapeWindow(WINDOW_SIZE_W, WINDOW_SIZE_H);
            break;
    }
}

//-----

void close(void)
{
    // stop the simulation
    simulationRunning = false;

    // wait for graphics and haptics loops to terminate
    while (!simulationFinished) { cSleepMs(100); }

    // close haptic device
    tool->stop();
}
```

This GLUT window mouse menu callbacks.
In this application we have simply implemented the option for setting the window to full screen mode and vice-versa.

example

```
void updateGraphics(void)
{
    // render world
    camera->renderView(displayW, displayH);

    // Swap buffers
    glutSwapBuffers();

    // check for any OpenGL errors
    GLenum err;
    err = glGetError();
    if (err != GL_NO_ERROR) printf("Error:  %s\n", gluErrorString(err));

    // inform the GLUT window to call updateGraphics again (next frame)
    if (simulationRunning)
    {
        glutPostRedisplay();
    }
}
```

Render the scene from the camera.
The result is displayed in the main
Open GL window.

Since OpenGL uses double buffering
to avoid image flickering, we swap
buffers just after having rendered the
last image frame

example

```
void updateHaptics(void)
{
    // main haptic simulation loop
    while(simulationRunning)
    {

        // compute global reference frames for each object
        world->computeGlobalPositions(true);

        // update position and orientation of tool
        tool->updatePose();

        // compute interaction forces
        tool->computeInteractionForces();

        // send forces to device
        tool->applyForces();
    }

    // exit haptics thread
    simulationFinished = true;
}
```

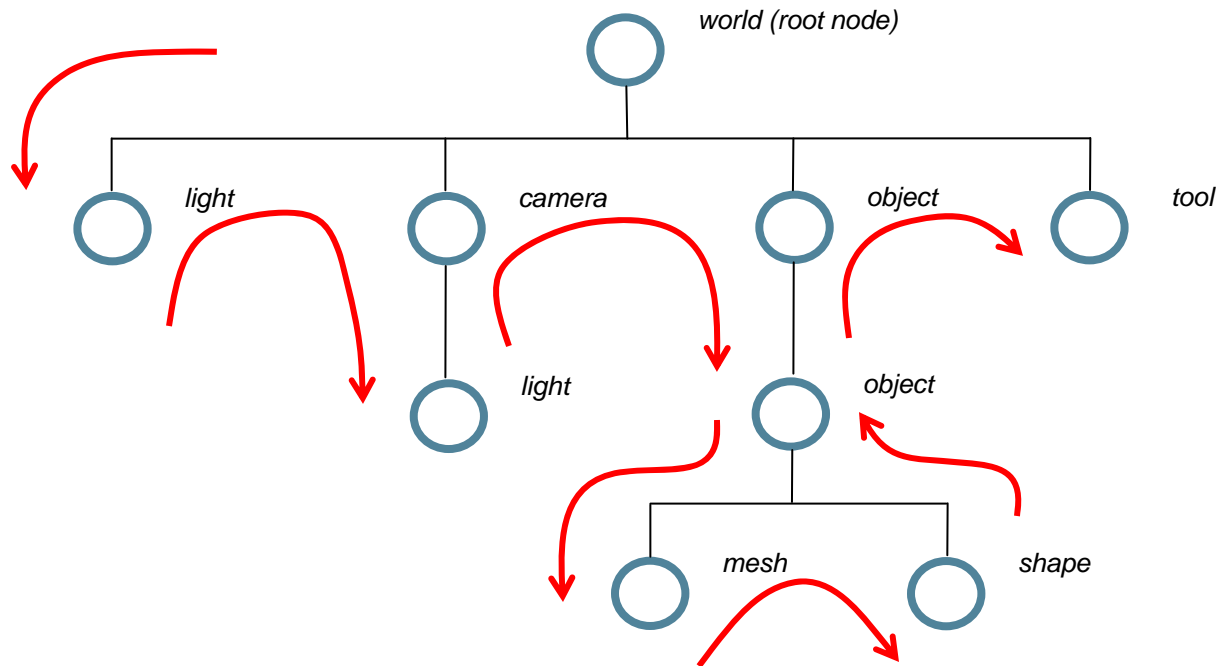
This is where our main haptics loop lives:

At each iteration we update the position of the tool by reading the new position of the haptic device, we then compute all collisions between the tool and the environment, and we apply the forces to the haptic device.

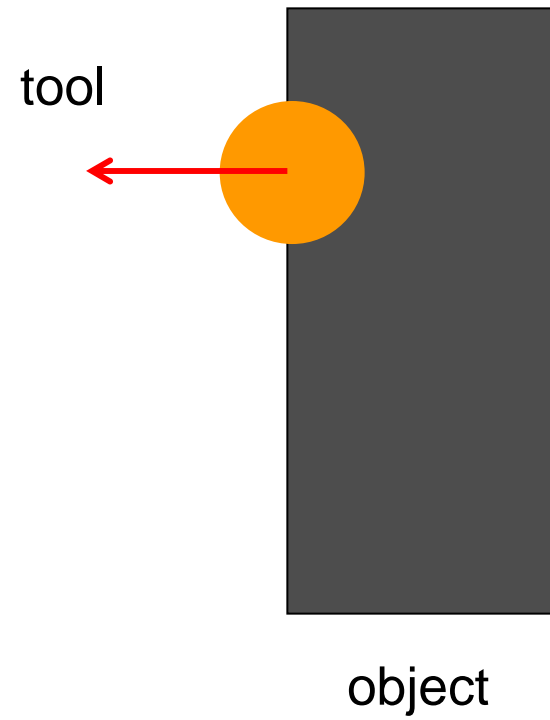
If our simulation contains dynamics modeling, this where we would typically update the equations of motion for each object.

See the other CHAI3D examples for more information.

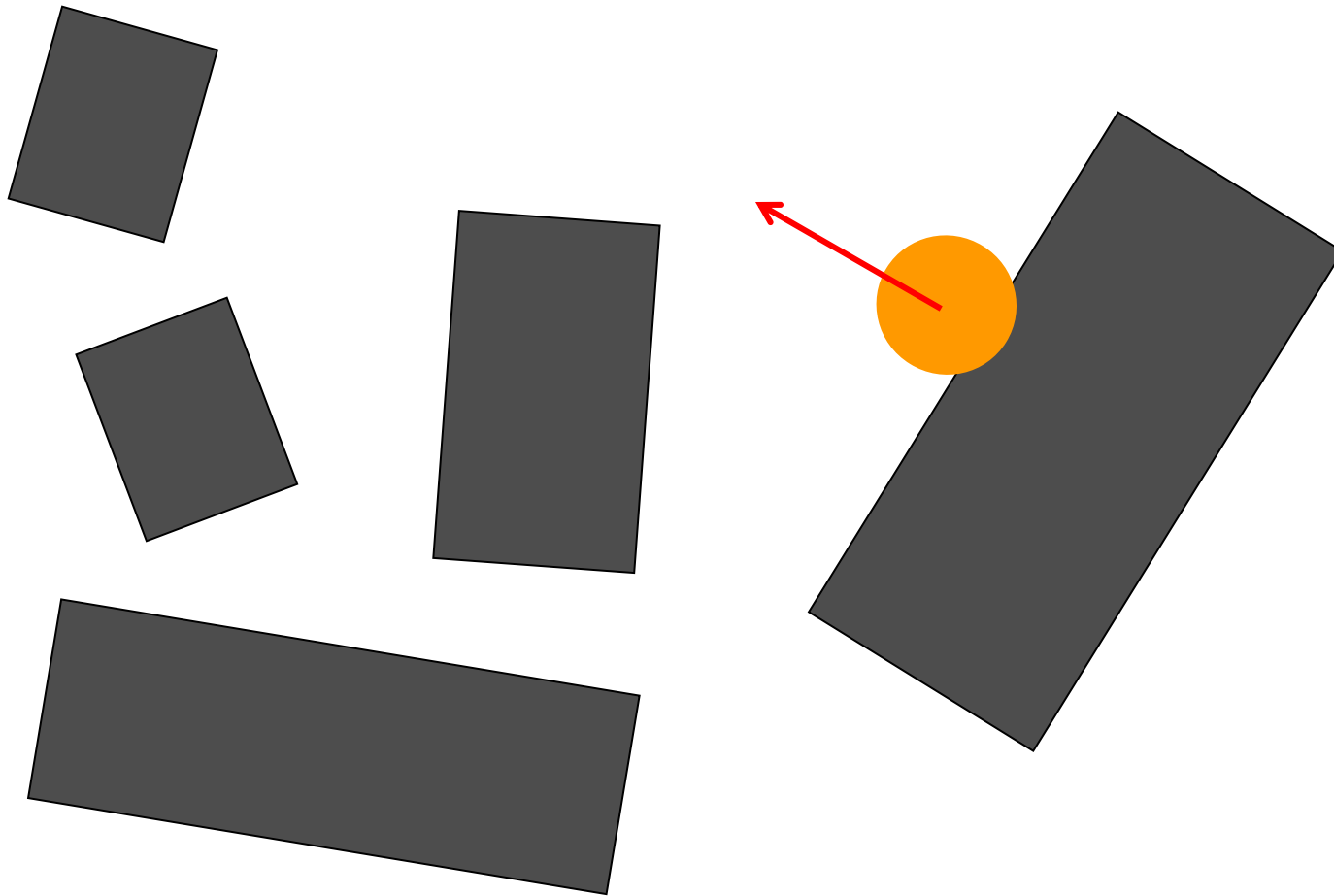
traversing the scene graph



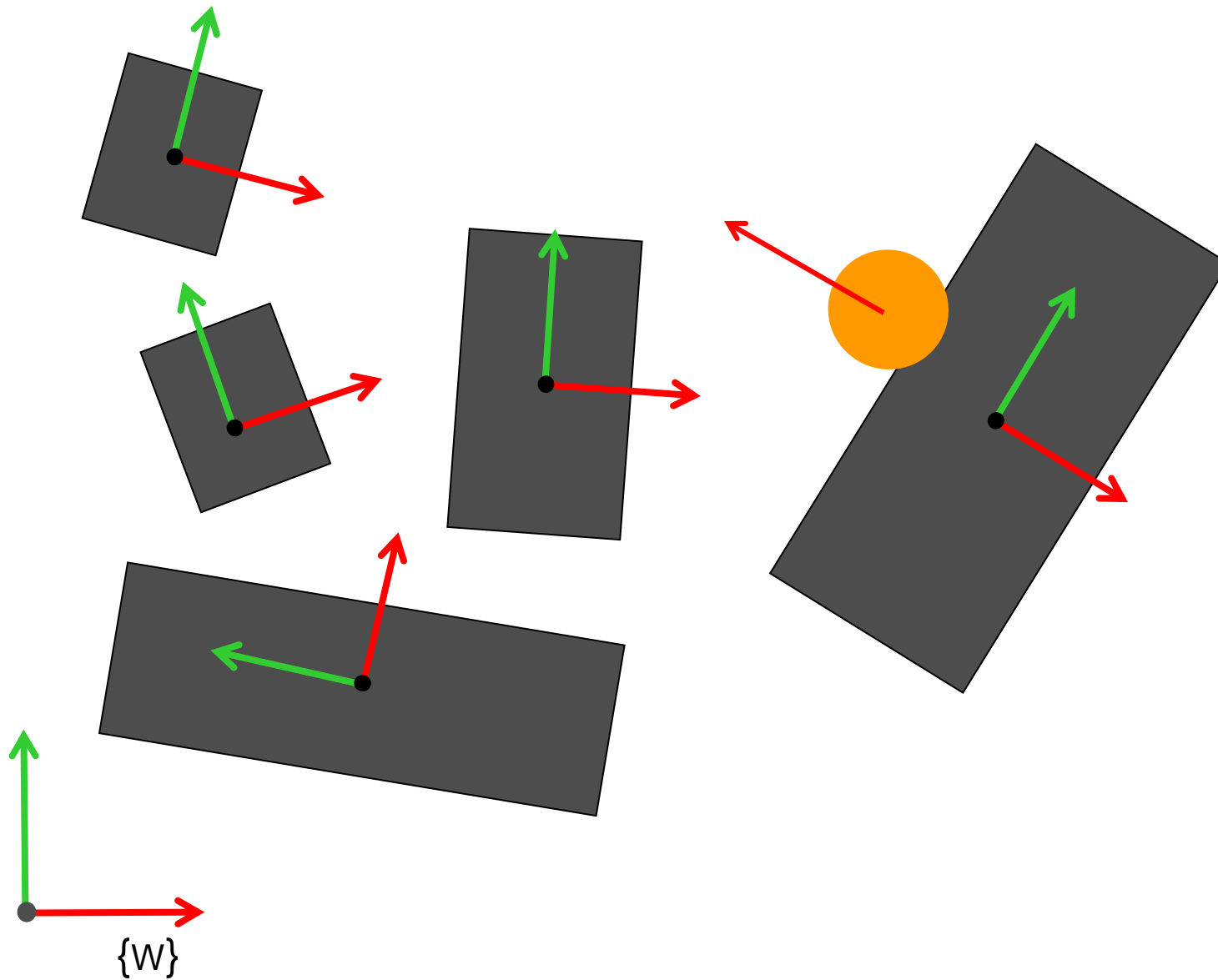
traversing the scene graph



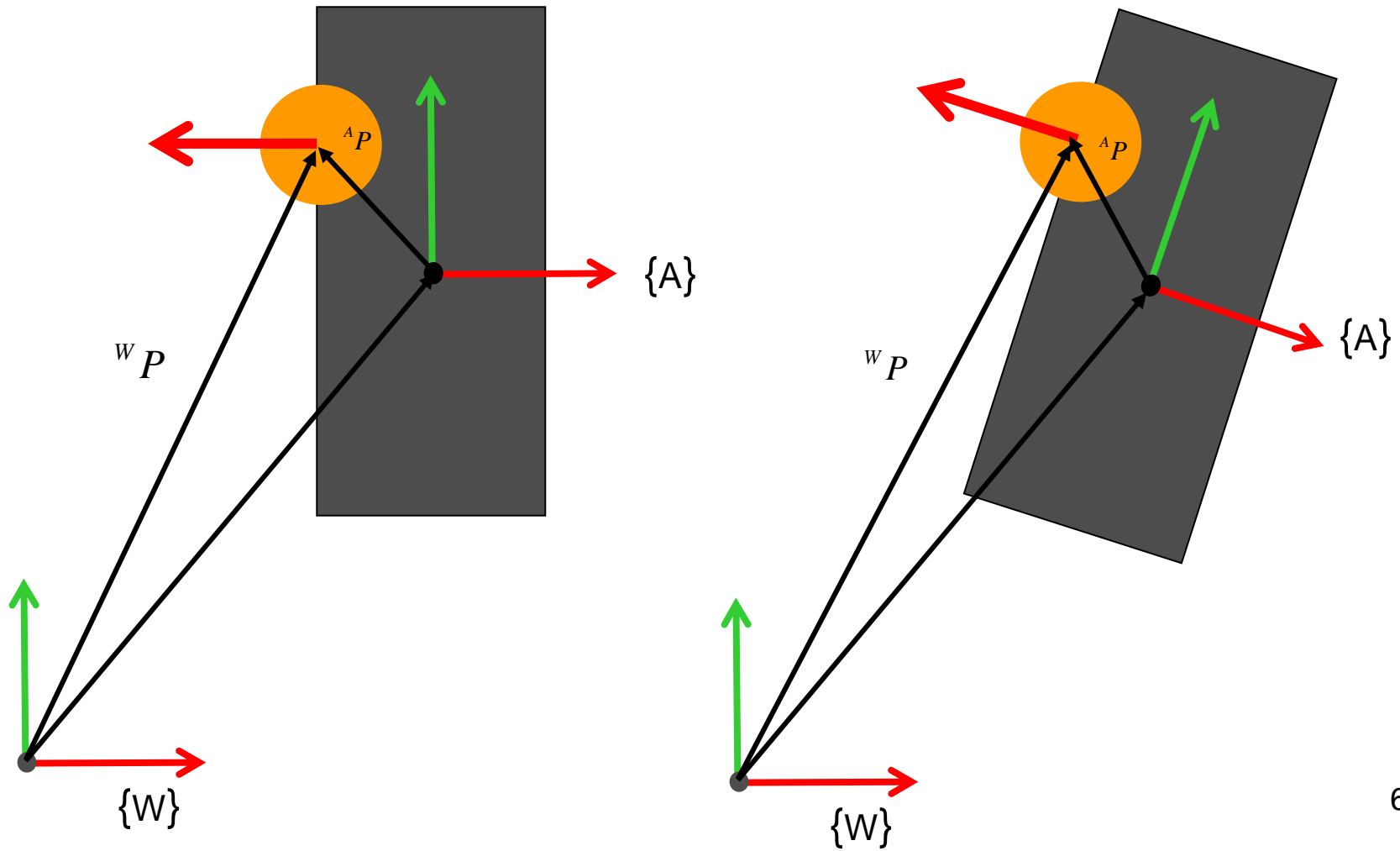
traversing the scene graph



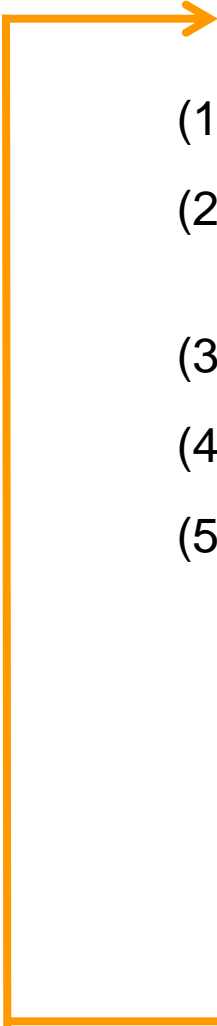
traversing the scene graph



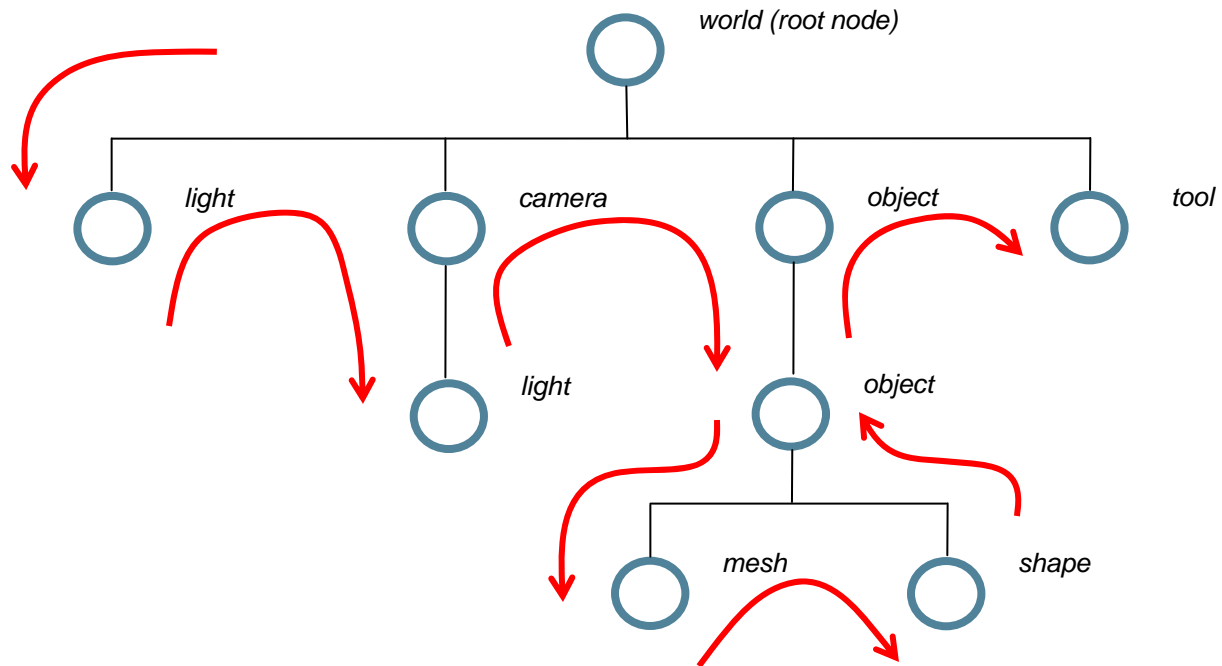
traversing the scene graph



traversing the scene graph

- 
- (1) Read the position of the haptic device
 - (2) For each object (potential field) in the environment, compute the position of the **tool** in relation to the **local reference frame**
 - (3) Compute the reaction force in the local frame
 - (4) Convert the reaction force in the world frame ${}^W F = {}^W R^A F$
 - (5) Send the force to the haptic device

traversing the scene graph



outline

- organization of CHAI3D
- haptic device software interface
- creating a virtual world
- scene graph
- building object primitives
- programming haptic effects
- example
- **developing your own primitives**

programming new primitives

```
class cShapeSphere : public cGenericObject
{
public:

    ///! Constructor of cShapeSphere.
    cShapeSphere(const double& a_radius);

    ///! Destructor of cSphere.
    virtual ~cShapeSphere() {};

    ///! Render object in OpenGL.
    virtual void render(const int a_renderMode=0);

    ///! Update bounding box of current object.
    virtual void updateBoundaryBox();

    ///! Object scaling.
    virtual void scaleObject(const cVector3d& a_scaleFactors);

    ///! Update the geometric relationship between the tool and the current object.
    virtual void computeLocalInteraction(const cVector3d& a_toolPos,
                                         const cVector3d& a_toolVel,
                                         const unsigned int a_IDN);

    ///! Set radius of sphere.
    void setRadius(double a_radius) { m_radius = cAbs(a_radius); updateBoundaryBox();}

    ///! Get radius of sphere.
    double getRadius() { return (m_radius); }

protected:

    ///! radius of sphere.
    double m_radius;
};
```

programming new primitives

```
class cShapeSphere : public cGenericObject
{
public:

    ///! Constructor of cShapeSphere.
    cShapeSphere(const double& a_radius);

    ///! Destructor of cSphere.
    virtual ~cShapeSphere() {};

    ///! Render object in OpenGL.
    virtual void render(const int a_renderMode=0);

    ///! Update bounding box of current object.
    virtual void updateBoundaryBox();

    ///! Object scaling.
    virtual void scaleObject(const cVector3d& a_scaleFactors);

    ///! Update the geometric relationship between the tool and the current object.
    virtual void computeLocalInteraction(const cVector3d& a_toolPos,
                                         const cVector3d& a_toolVel,
                                         const unsigned int a_IDN);

    ///! Set radius of sphere.
    void setRadius(double a_radius) { m_radius = cAbs(a_radius); updateBoundaryBox();}

    ///! Get radius of sphere.
    double getRadius() { return (m_radius); }

protected:

    ///! radius of sphere.
    double m_radius;
};
```

programming new primitives

```
void cShapeSphere::render(const int a_renderMode)
{
    // render material properties
    if (m_useMaterialProperty)
    {
        m_material.render();
    }

    // allocate a new OpenGL quadric object for rendering a sphere
    GLUQuadricObj *sphere;
    sphere = gluNewQuadric ();

    // set rendering style
    gluQuadricDrawStyle (sphere, GLU_FILL);

    // set normal-rendering mode
    gluQuadricNormals (sphere, GLU_SMOOTH);

    // render texture property if defined
    if ((m_texture != NULL) && (m_useTextureMapping))
    {
        m_texture->render();

        // generate texture coordinates
        gluQuadricTexture(sphere, GL_TRUE);

        // turn off texture rendering
        glDisable(GL_TEXTURE_2D);
    }

    // render a sphere
    gluSphere(sphere, m_radius, 36, 36);

    // delete our quadric object
    gluDeleteQuadric(sphere);
}
```

programming new primitives

```
class cShapeSphere : public cGenericObject
{
public:

    ///! Constructor of cShapeSphere.
    cShapeSphere(const double& a_radius);

    ///! Destructor of cSphere.
    virtual ~cShapeSphere() {};

    ///! Render object in OpenGL.
    virtual void render(const int a_renderMode=0);

    ///! Update bounding box of current object.
    virtual void updateBoundaryBox();

    ///! Object scaling.
    virtual void scaleObject(const cVector3d& a_scaleFactors);

    ///! Update the geometric relationship between the tool and the current object.
    virtual void computeLocalInteraction(const cVector3d& a_toolPos,
                                         const cVector3d& a_toolVel,
                                         const unsigned int a_IDN);

    ///! Set radius of sphere.
    void setRadius(double a_radius) { m_radius = cAbs(a_radius); updateBoundaryBox();}

    ///! Get radius of sphere.
    double getRadius() { return (m_radius); }

protected:

    ///! radius of sphere.
    double m_radius;
};
```

programming new primitives

```
void cShapeSphere::updateBoundingBox()  
{  
    m_boundaryBoxMin.set(-m_radius, -m_radius, -m_radius);  
    m_boundaryBoxMax.set( m_radius,  m_radius,  m_radius);  
}  
  
void cShapeSphere::scaleObject(const cVector3d& a_scaleFactors)  
{  
    m_radius = a_scaleFactors.x * m_radius;  
}
```

programming new primitives

```
class cShapeSphere : public cGenericObject
{
public:

    ///! Constructor of cShapeSphere.
    cShapeSphere(const double& a_radius);

    ///! Destructor of cSphere.
    virtual ~cShapeSphere() {};

    ///! Render object in OpenGL.
    virtual void render(const int a_renderMode=0);

    ///! Update bounding box of current object.
    virtual void updateBoundaryBox();

    ///! Object scaling.
    virtual void scaleObject(const cVector3d& a_scaleFactors);

    ///! Update the geometric relationship between the tool and the current object.
    virtual void computeLocalInteraction(const cVector3d& a_toolPos,
                                         const cVector3d& a_toolVel,
                                         const unsigned int a_IDN);

    ///! Set radius of sphere.
    void setRadius(double a_radius) { m_radius = cAbs(a_radius); updateBoundaryBox();}

    ///! Get radius of sphere.
    double getRadius() { return (m_radius); }

protected:

    ///! radius of sphere.
    double m_radius;
};
```

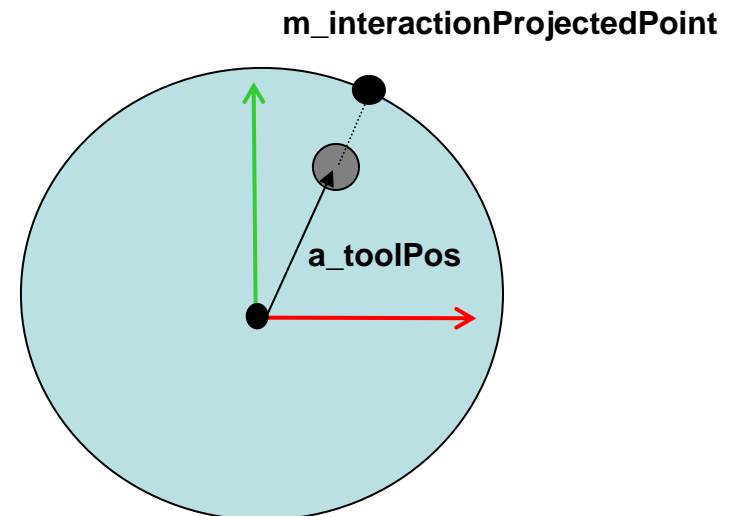
programming new primitives

```
void cShapeSphere::computeLocalInteraction(const cVector3d& a_toolPos,
                                           const cVector3d& a_toolVel,
                                           const unsigned int a_IDN)
{
    // compute distance from center of sphere to tool
    double distance = a_toolPos.length();

    // from the position of the tool, search for the nearest point located
    // on the surface of the sphere
    if (distance > 0)
    {
        m_interactionProjectedPoint = cMul( (m_radius/distance), a_toolPos);
    }
    else
    {
        m_interactionProjectedPoint = a_toolPos;
    }

    // check if tool is located inside or outside of the sphere
    if (distance <= m_radius)
    {
        m_interactionInside = true;
    }
    else
    {
        m_interactionInside = false;
    }
}
```

m_interactionInside = true
tool is located inside object

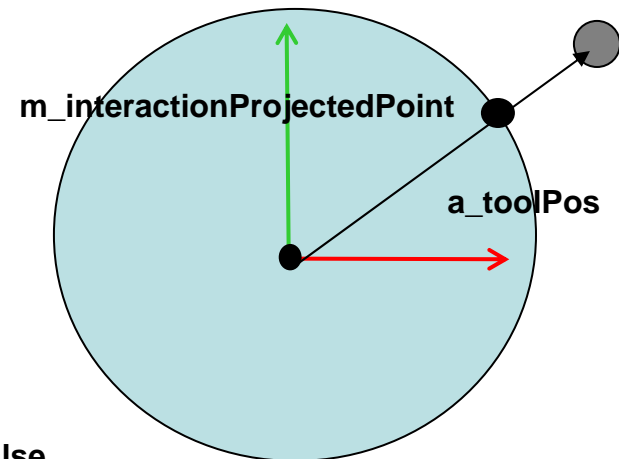


programming new primitives

```
void cShapeSphere::computeLocalInteraction(const cVector3d& a_toolPos,
                                           const cVector3d& a_toolVel,
                                           const unsigned int a_IDN)
{
    // compute distance from center of sphere to tool
    double distance = a_toolPos.length();

    // from the position of the tool, search for the nearest point located
    // on the surface of the sphere
    if (distance > 0)
    {
        m_interactionProjectedPoint = cMul( (m_radius/distance), a_toolPos);
    }
    else
    {
        m_interactionProjectedPoint = a_toolPos;
    }

    // check if tool is located inside or outside of the sphere
    if (distance <= m_radius)
    {
        m_interactionInside = true;
    }
    else
    {
        m_interactionInside = false;
    }
}
```



m_interactionInside = false
tool is located outside object