Algoryx®

AGX DYNAMICS

API PROGRAMMING COURSE

Algoryx®
AGX DYNAMICS

Content

- Section 1 - Physics engine fundamentals
- Section 2 - Physics and math behind AGX
- Section 3 - AGX C++ API
- Section 4 - API Usage
- Section 5 - Brief overview of other parts of the API
- Appendix 1 - Lua programming with AGX
- Appendix 2 - Python programming with AGX
- Exercises - Practical exercises using C#/Lua/C++ (not part of main document)

Other documentation material

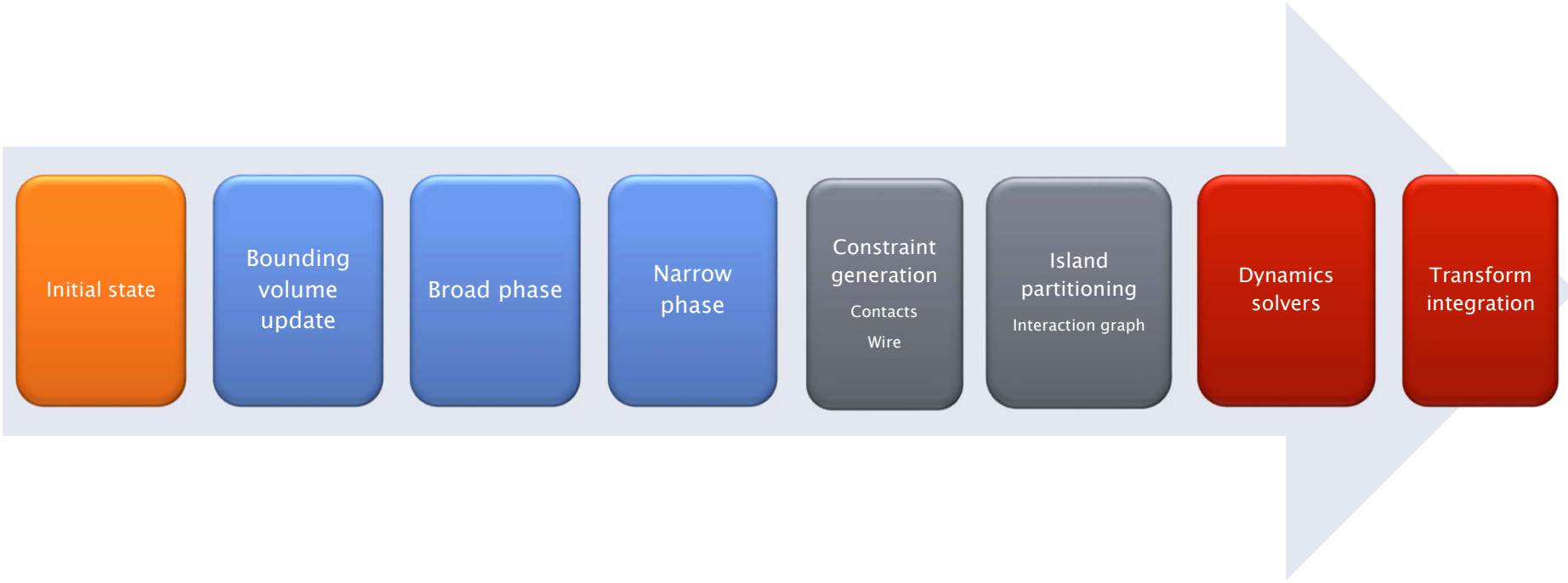
- *Except for this presentation*, have a look at
- AGX Dynamics User Guide (pdf)
- AGX Dynamics API documentation (Doxygen)
- Lua tutorials
- Lua Wire tutorials
- C++ Tutorials
- Python Tutorials

All can be reached from AGX Dynamics main page after installation.

Section 1

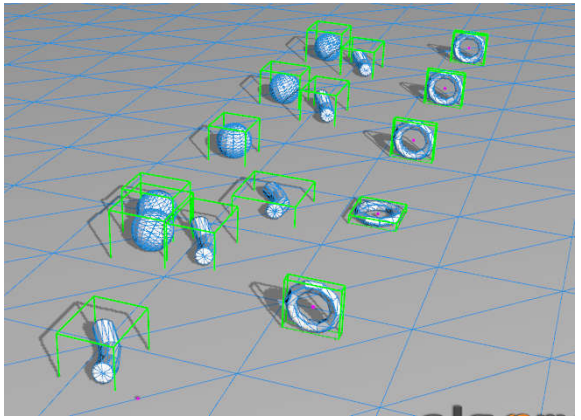
Physics engine fundamentals

The pipeline

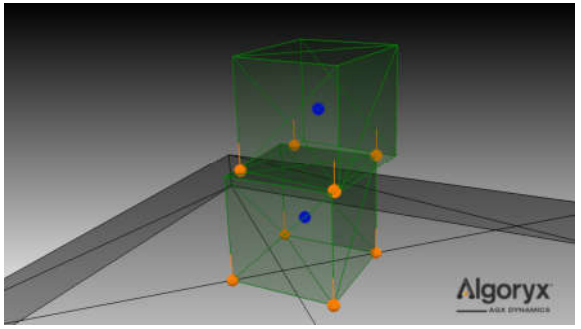


Collision detection

- Broad Phase
  - Do bounding volumes overlap?



- Narrow Phase
  - Test geometry overlap, extract contact data:
    - Point
    - Normal
    - Depth
  - Performed in so called *colliders.*

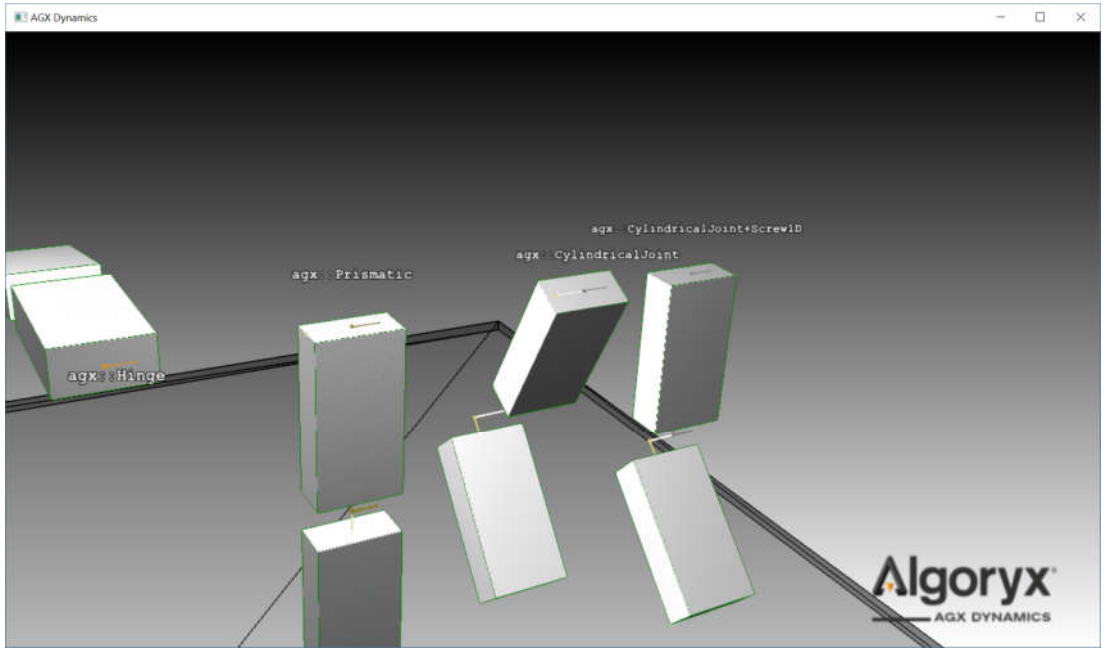


Constraints

Conditions that limit the velocity or position of one or more bodies, relative to each other or relative to the world.

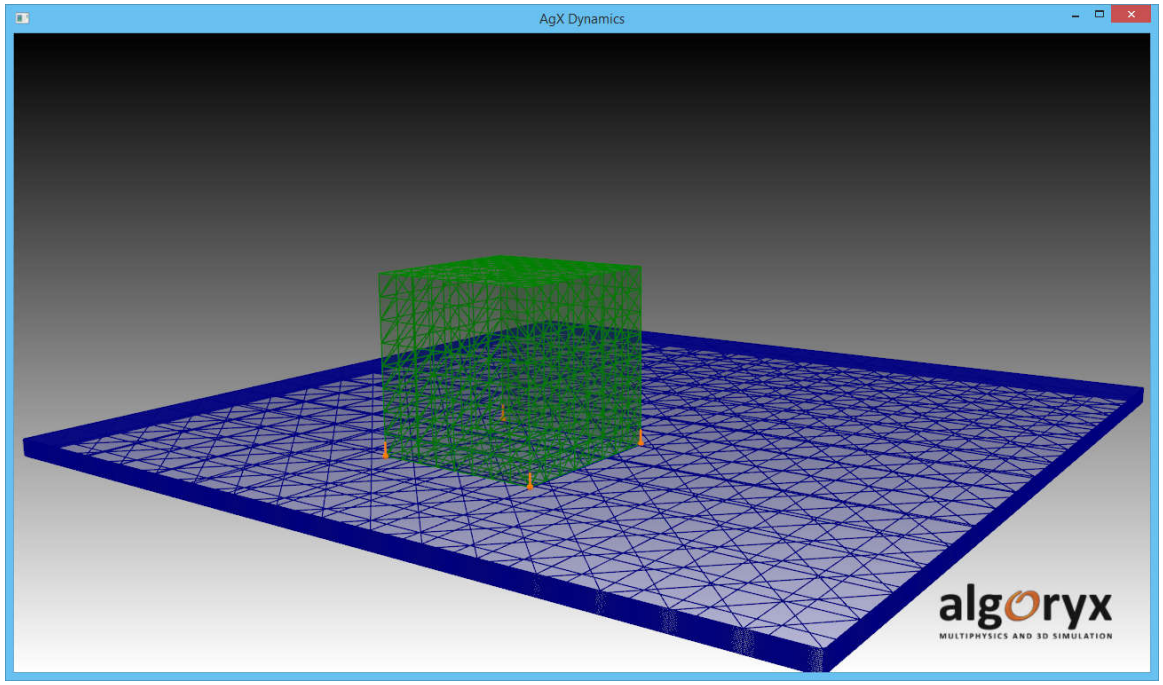
AGX provides a selection of types:

- Hinge
- Prismatic
- Cylindrical
- LockJoint



Constraints from Contacts

- Relation between coordinates and momenta.
- A condition a/the *solver* will make sure is fulfilled.
  - Example: A box resting on a plane has four contact points – *one* solution; each contact point holding up a quarter of the weight of the box.

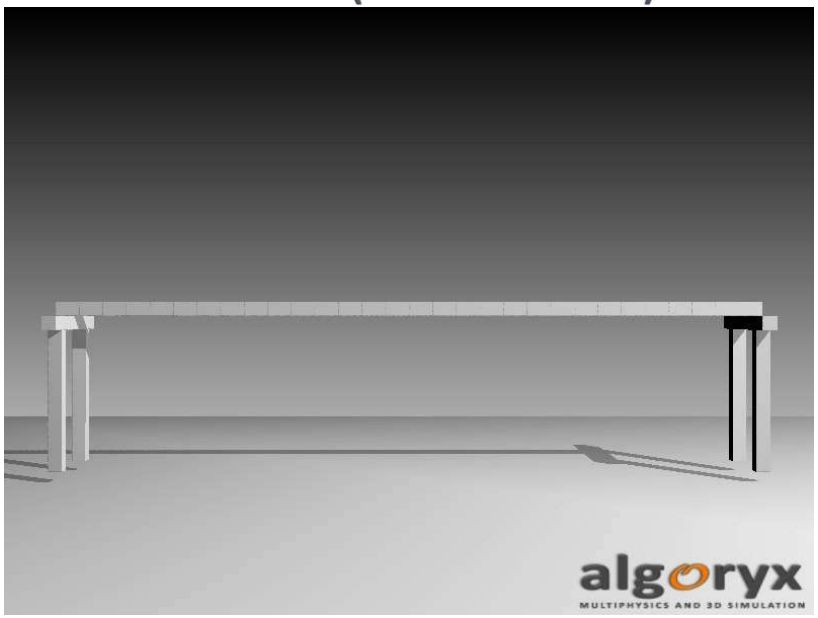


Solvers

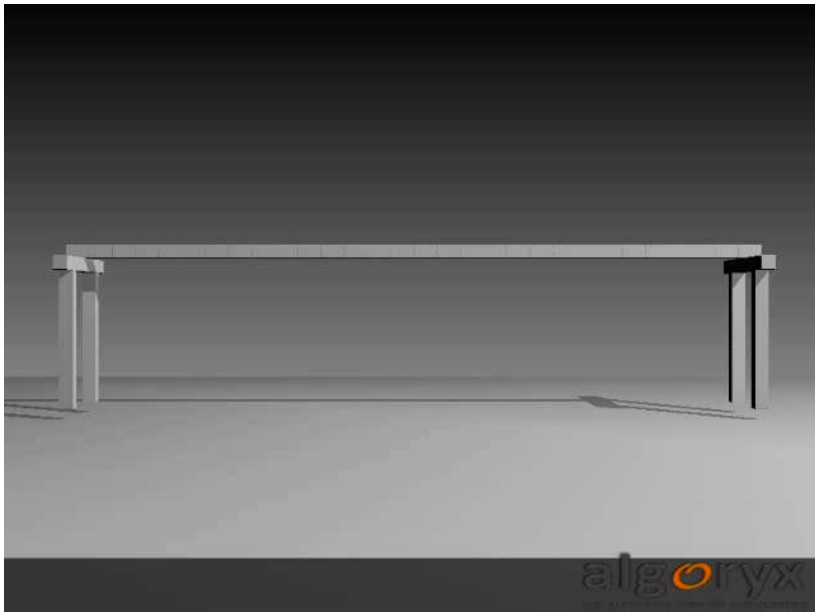
- Direct solver
  - Dense, sparse
  - ”Exact” (machine precision @ linear solves)
- Iterative solver
  - Gauss-Seidel, Jacobi, Conjugate Gradient
  - Achieve plausible results, fast
- AGX Approach: Hybrid
  - Combination of direct and iterative algorithms
    - Highly configurable
  - Best of two worlds.

Comparison: Direct v.s Iterative solver

Iterative solver (Gauss Seidel)



Direct solver



- Iterative solver issues
  - Stiff systems
  - Large mass ratios (1/100)
  - “Sloppy”

Section 2

PHYSICS AND MATH BEHIND AGX

Constraint based physics

- **Advantages with constraint based physics in AGX**
  - Stiff interactions.
  - Stable at large time steps (fundamental for real-time).
  - System is a block sparse linear system, that can be solved efficiently.
  - Consistent multiphysics coupling, e.g., rigid body contacts, wires, hydraulics, granulars.
  - Constraints can be created and/or deleted on the fly (or activated/deactivated).

- **Constraints have real physics based modeling parameters!**
  - Controlled elasticity and dissipation.

Constraint formulation

A constraint is satisfied when the constraint function *g* is zero:

$$g(q) = 0$$

where *q* are the variables of the system – e.g., positions and orientations. If *g* is stationary, so is its time derivative:

$$\dot{g}(q) = \frac{d}{dt} g(q) = \frac{\partial g}{\partial q} \frac{dq}{dt} = Gv = 0$$

where *G* is the Jacobian of the constraints and *v* the velocities (e.g., linear and angular).

Constraint formulation

$Gv = 0$ defines "allowed" velocities along the so called constraint surface, defined by the Jacobian. Velocities violating the constraint surface are projected back onto the surface.

However, only a mathematical constraint strictly enforces *g = 0*. A physics based constraint has elasticity. A potential function that models this:

$$U(q) = \frac{1}{2\epsilon} \|g(q)\|^2$$

with a corresponding (constraint) force:

$$f_c = -\frac{\partial U}{\partial q} = -\frac{1}{\epsilon} G^T g(q)$$

Constraint formulation

Let:

$$\lambda = -\frac{1}{\epsilon} g(q)$$

and introduce a damping term $\gamma \dot{g}(q)$ – since physics-based constraints have non-zero constraint velocity, associated with dissipation – to receive our final and complete equations of motion:

$$M\ddot{q} = f_e + G^T \lambda$$

$$\epsilon\lambda + g(q) + \gamma \dot{g}(q) = 0$$

The parameter ϵ is associated with the constraint **compliance** in the AGX API.
The parameter γ is associated with the constraint **damping** in the AGX API.

Constraint formulation

$$M\ddot{q} = f^e + G^T\lambda$$
$$\epsilon\lambda + g(q) + \gamma\dot{g}(q) = 0$$

These equations of motion are then discretized in time, with a time step *h*, via a discrete variational principle with the result:

$$\begin{bmatrix} M & -G_k^T \\ G_k & \Sigma \end{bmatrix} \begin{bmatrix} v_{k+1} \\ \lambda \end{bmatrix} = \begin{bmatrix} Mv_k + hf_k^e \\ -\dfrac{4}{h\left(1+4\dfrac{\gamma}{h}\right)} g_k + \dfrac{1}{1+4\dfrac{\gamma}{h}} G_k v_k \end{bmatrix}$$

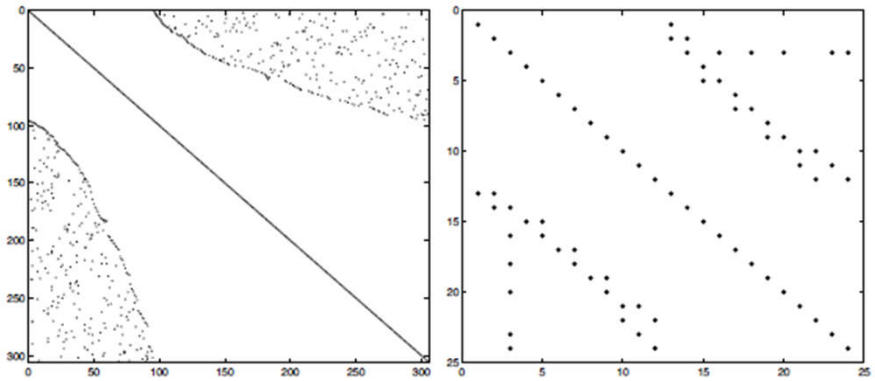
Constraint formulation

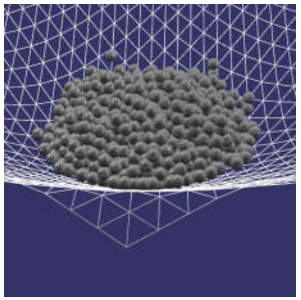
$$\begin{bmatrix} M & -G_k^T \\ G_k & \Sigma \end{bmatrix} \begin{bmatrix} v_{k+1} \\ \lambda \end{bmatrix} = \begin{bmatrix} Mv_k + hf_k^e \\ -\dfrac{4}{h\left(1+4\dfrac{\gamma}{h}\right)} g_k + \dfrac{1}{1+4\dfrac{\gamma}{h}} G_k v_k \end{bmatrix}$$

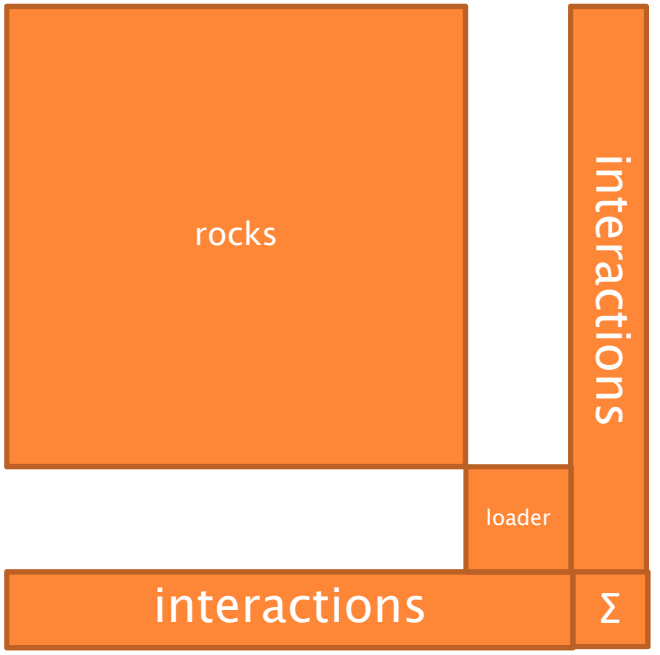
Σ correspond to regularized masses of the constraints, and

- Reintroduce elasticity in the constraint formulation
- Dramatically improve solvability of the linear system
- Together with damping terms provide a low pass filter for stable time stepping

Solvers







- Pile of rocks (2 000x2 000)
  - Gauss-Seidel
- Wheel loader
  - Direct solver (in-house)

References

Selected references and further reading

Ghosts and machines: regularized variational methods for interactive simulations of multibodies with dry frictional contacts. Lacoursière, Claude, Umeå universitet (2007). http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-1143

Hybrid, multi-resolution wires with massless frictional contacts, M. Servin, C. Lacoursière, K. Bodin, IEEE Transactions on Visualization and Computer Graphics, Volume: 17 Issue:7, On page(s): 970 - 982, July (2011). IEEE computer Society Digital Library. IEEE Computer Society,

Examining the smooth and nonsmooth discrete element approaches to granular matter, M. Servin, C. Lacoursère, D. Wang, K. Bodin, Particles 2011 – ECCOMAS International Conference on Particle-based Methods, Barcelona (2011). abstract slides

Outlet design optimization based on large-scale nonsmooth DEM simulation, D. Wang, M. Servin, K. Mickelsson, Particles 2011 – ECCOMAS International Conference on Particle-based Methods, Barcelona (2011).abstract

Constraint based particle fluids on GPGPU, K. Bodin, C. Lacoursière, M. Nilsson, M. Servin, Particles 2011 – ECCOMAS International Conference on Particle-based Methods, Barcelona (2011). abstract

Regularized multibody dynamics with dry frictional contacts, C. Lacoursière and M. Servin, Euromech Colloquium: Nonsmooth contact and impact laws in mechanics, July 6th - 8th 2011, Grenoble, France, (2011) pdf, web

Constraint fluids, K. Bodin, C. Lacoursière, M. Servin, IEEE Transactions on Visualization and Computer Graphics, Vol pp, Issue 99. 2011. IEEE computer Society Digital Library. IEEE Computer Society, <http://doi.ieeecomputersociety.org/10.1109/TVCG.2011.29>

Interactive simulation of elastic deformable materials, by M. Servin, C. Lacoursière and N. Melin, In Proceedings of SIGRAD Conference 2006 in Skövde, Sweden, Linköping University Electronic Press, Linköping, 22-32 (2006)

A regularized time stepper for multibody systems, C. Lacoursière. In J. Sporring, K. Erleben, and H. Dohlmann, editors, PDE Methods in Computer Graphics. Charles River Media, 2005.

Regularized, stabilized, variational methods for multibodies, C. Lacoursière, In Dag Fritzson Peter Bunus and Claus Führer, editors, The 48th Scandinavian Conference on Simulation and Modeling (SIMS 2007), 30-31 October, 2007, Göteborg (Särö), Sweden, Linköping Electronic Conference Proceedings, pages 40–48. Linköping University Electronic Press, December 2007.

A parallel block iterative method for interactive contacting rigid multibody simulations on multicore PCs, C. Lacoursière. A, In PARA'06, pages 956–965, 2006.

Visual Simulation of Machine Concepts for Forest Biomass Harvesting, M. Servin, A. Backman, K. Bodin, U. Bergsten, D. Bergström, B. Löfgren, T. Nordfjell, I. Wästerlund, VRIC 2008 - 10th International Conference on Virtual Reality (Laval Virtual), (2008).

Section 3

AGX C++ API

Getting started

- Topics
  - Build configuration
  - Check return values
  - Structure
  - Memory and Reference pointers
  - Simulation
  - Hello world
    - init/shutdown
  - Threads
  - agxViewer
    - Arguments, keys
  - Parallelization
  - Coupling to external simulation

Build configuration

- AGX flavours
  - Windows
    - Compiler version: Visual Studio 2013/2015
    - Platform: x86, x64
    - Real precision: double
  - Linux
    - gcc, clang
  - Mac
    - gcc, clang
- AGX is built as dynamic libraries (.dll/.so)
- If python is desired: 64 bit only, version >= 3.5 (good to be installed before AGX installation).

Visual Studio Settings

- Properties ➡ C/C++ ➡ Code Generation
  - MultiThreaded DLL (/MD)
- Properties ➡ C/C++ ➡ General ➡ Additional include directories
  - <agx-dir>/include
  - <agx-dir>/include/1.4
- Properties ➡ Linker ➡ General ➡ Additional library directories
  - <agx-dir>/lib/x86 or
  - <agx-dir>/lib/x64
- Important to not mix debug/release libraries (d-suffix)
  - agxPhysics.lib (release)
  - agxPhysicsd.lib (debug)
- Mandatory libraries:
  - agxCore.lib, agxPhysics.lib, OpenThreads.lib

CMake

- AGX uses CMake for configuration
  - Generates project files from configuration files.
  - Optional.
  - Tutorials can be built using cmake/or by hand.
  - Look into the AGX User Manual for more information.

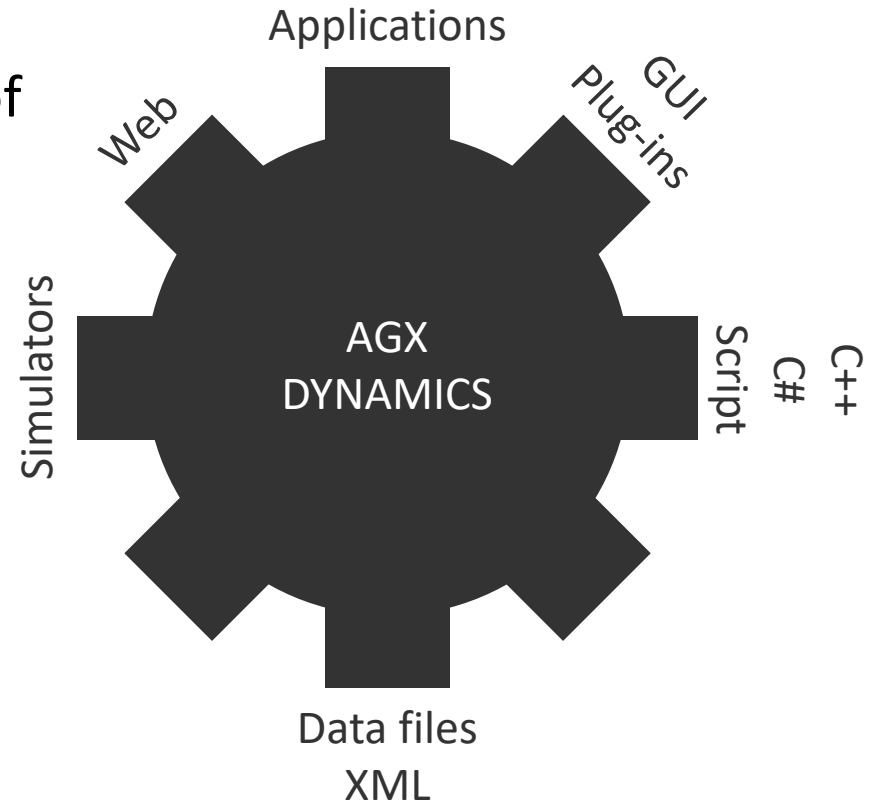
Check return values

- Most of the API calls in AGX return a value
  - bool
  - pointers
- Make sure to check return values!
- No 100% guaranteed consistency checks in the whole API.
- AGX can throw exceptions. Use try/catch clause at the topmost level.
  - std::exception

```
try {
  // Initialize and run simulation.
}
catch ( const std::exception& e ) {
  std::cerr << "Caught exception: " << e.what() << std::endl;
}
```

Interfacing AGX

- Accessing AGX
  - Written in C++, official API.
  - Uses SWIG to export a majority of the API to C# .NET, or Java.
  - Uses Python or Lua for testing, demonstrating, prototyping and illustrating the API.



Namespaces - libraries

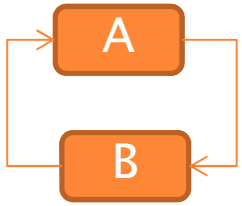
Namespace	Library name	Description
agxSabre	agxSabre.lib	Block sparse matrix solver.
agx, agxData	agxCore.lib	Core dynamics library, including basic data types: Vec3, Quat, Matrices ref_ptr, etc.
agxCollide	agxPhysics.lib	Geometric intersection system for calculating intersection points, normals, penetration depth. Classes for various primitives such as Sphere, Cylinder, Box, ...
agxIO	agxPhysics.lib	Contain classes for reading / (writing) various file formats (images, models).
agxSDK	agxPhysics.lib	Simulation framework. Higher level abstraction for modeling a system, stepping collision and dynamics forward in discrete time steps. Event handling, materials.
agxStream	agxPhysics.lib	Serialization classes.
agxUtil	agxPhysics.lib	Various utility classes and functions.
agxNet	agxPhysics.lib	Classes for communication, sockets, compression.
agxOSG	agxOSG.lib	Various classes for integrating AGX to OpenSceneGraph. For testing/debug purposes.
agxModel	agxModel.lib	High level modeling primitives, Tree, Terrain etc.
agxWire	agxPhysics.lib	Wire, winches, connection links etc.
agxCable	agxCable.lib	Cables for modeling robotic cables etc.
agxDriveTrain	agxModel.lib	Clutches, Gears, Engine etc.
agxHydraulics	agxHydraulics.lib	Valves, pumps, pipes etc.

Memory and reference pointers

- Most objects in the user API are derived from agx::Referenced
  - Intrusive reference counting, slightly different usage than std::shared_ptr
  - Cannot be created on the stack (protected destructors).
  - Keeps track on references.
  - Automatically deallocated at last dereference.

```
{ // Scope begin.
  // Create reference pointer object to a rigid body.
  agx::ref_ptr< agx::RigidBody > rb1 = new agx::RigidBody();

  // Most types have typedefs for this: "Object" + Ref
  agx::RigidBodyRef rb2 = new agx::RigidBody();

  // De-referenced, object will be deleted.
  rb1 = nullptr;
} // Scope end. rb2 leaves scope and will be deleted.
```

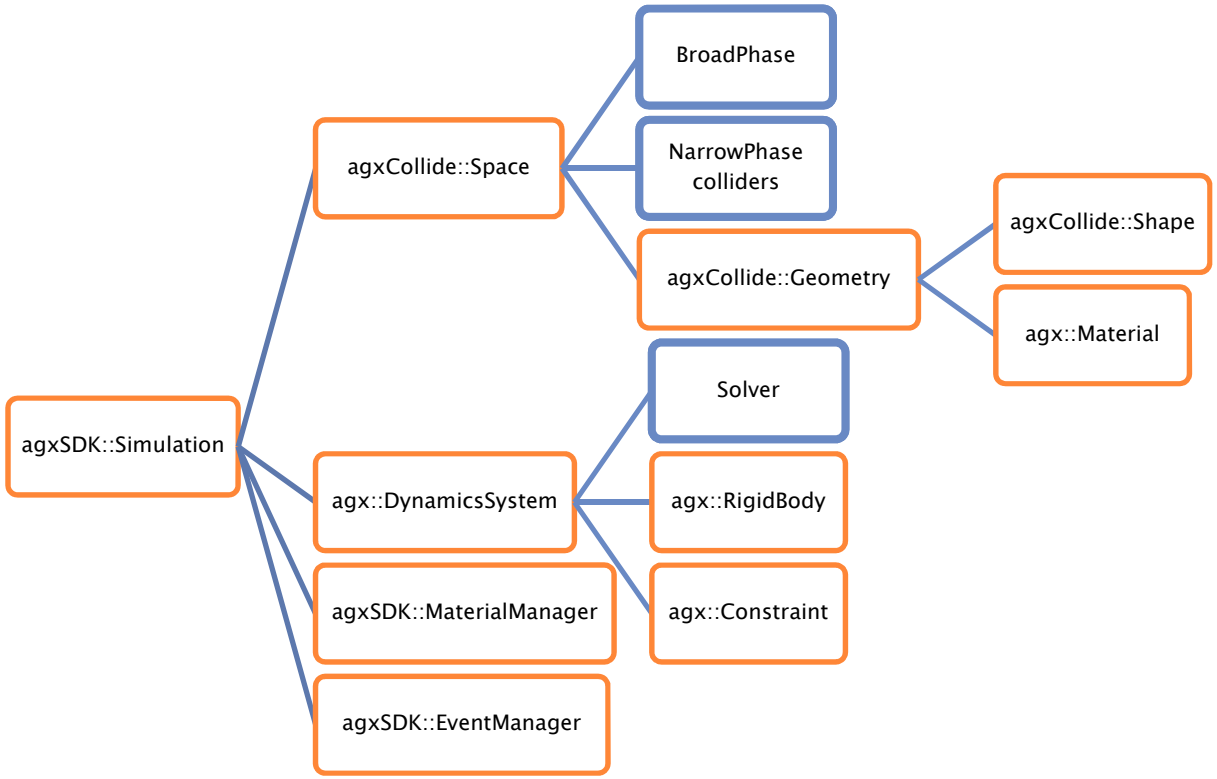
- Potential problem: Circular references
  - A references B, B references A -> A and B will never be deallocated.
- Solution
  - agx::observer_ptr: Will be set to 0 when object is deallocated.




```
// Reference pointer to a rigid body.
agx::RigidBodyRef rb = new agx::RigidBody();

// Observer pointer to the rigid body.
agx::RigidBodyObserver rbObserver = rb;

if ( rbObserver != nullptr )
  ; // Observer is valid, will go here.

// Delete rb.
rb = nullptr;

if (rbObserver != nullptr)
  ; // The observer is now a null pointer, will NOT go here.
```

agxSDK::Simulation

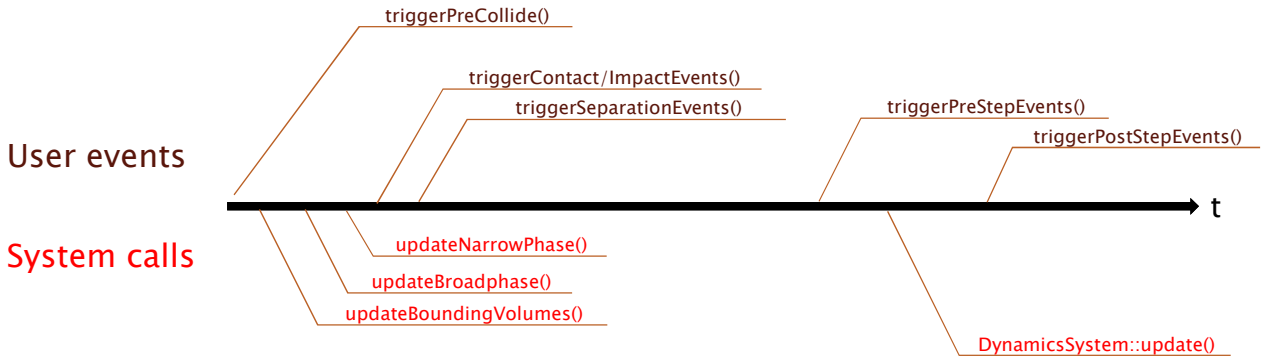
- Master of a simulation.
- Keep references to fundamental objects.
- Responsible for stepping the system.
- Interface for adding/removing all parts of a simulation.
- agxCollide::Space
  - Perform collision detection
- agx::DynamicsSystem
  - Solver, Integrator



Stepping time

- Stepping a system:
  - agxSDK::Simulation::stepForward() – take *one* time step forward in time.
  - agxSDK::Simulation::stepTo(agx::Real to) – take numerous time steps until time *to* is reached.
- Setting time step (dt)

```
agxSDK::SimulationRef simulation = new agxSDK::Simulation();

// Set time step length (0.016667 s), 60 Hz.
simulation->setTimeStep( agx::Real( 1.0 / 60.0 ) );

// Step the simulation 0.016667 s forward in time.
simulation->stepForward();

// Step the simulation 2 seconds forward from current time.
simulation->stepTo( simulation->getTimeStamp() + agx::Real( 2 ) );
```



Schematic view of the simulation time line

Hello World

- agxIO::Environment
  - Locating license file
  - Plugins
  - Kernels (.agxKernels)
  - Entities (.agxEntity)
  - Settings (settings.cfg/*.schema)
- agx::init()
  - Initialize AGX before (almost any) call to the API.
  - Will initialize internal system for types, kernels, plugins etc.
- agx::shutdown()
  - Will tear down what's initialized in agx::init().
  - Should be the last call to the AGX API.
- agx::AutoInit
  - Scope automatic initialization and shutdown.

```
void main(void)
{
  agx::AutoInit autoInit;
  agx::RigidBodyRef helloWorld = new agx::RigidBody( "Hello world!" );
  std::cout << helloWorld->getName() << std::endl;
}
```

Hello World

```
int main( int /*argc*/, char** /*argv*/ )
{
  // Before initializing AGX we've to specify where the mandatory
  // resources are located.
  const agx::String agxPath = "C:/Path/To/AGX-version/";
  const agx::String licensePath = "C:/MyLicenses/";

  // AGX searches for the license file in the resources directories.
  AGX_ENVIRONMENT().getFilePath( agxIO::Environment::RESOURCE_PATH ).pushbackPath( licensePath );

  // Mandatory plugins directory is located in the bin folder.
  AGX_ENVIRONMENT().getFilePath( agxIO::Environment::RESOURCE_PATH ).pushbackPath( agxPath + "bin/x64/plugins" );
  AGX_ENVIRONMENT().getFilePath( agxIO::Environment::RUNTIME_PATH ).pushbackPath( agxPath + "bin/x64/plugins" );

  // Default settings for AGX and schema files are located
  // in the cfg directory in the data folder.
  AGX_ENVIRONMENT().getFilePath( agxIO::Environment::RESOURCE_PATH ).pushbackPath( agxPath + "data/cfg" );

  // Initialize AGX.
  agx::AutoInit autoInit; // Will uninitialize AGX when leaving scope.

  try {
    // Create simulation, add a rigid body, simulate.
    agxSDK::SimulationRef simulation = new agxSDK::Simulation();
    simulation->add( new agx::RigidBody() );
    simulation->stepForward();
  }
  catch ( const std::exception& e ) {
    std::cerr << "Caught exception: " << e.what() << std::endl;
    return 1;
  }

  return 0;
}
```

Threads

- AGX supports multi threading
- The API is however NOT thread safe.
  - Can't manipulate objects from multiple threads, simultaneously.
- The thread in which init is called, is *main thread.*
- *All callbacks from Simulation::stepForward() must be done from the main* thread!

```
static void
agx::Thread::makeCurrentThreadMainThread();
```

- Each thread calling AGX must be made into an AGX thread using `agx::Thread::registerAsAgXThread();`
- Can have several instances of agxSDK::Simulation running in parallel.
  - With some (current) limitations
    - Creating/destroying wires
    - Using Lua with ExampleApplication (e.g. agxViewer).
    - …

```
void doStuffInParallel()
{
  // Create a simulation for this thread.
  agxSDK::SimulationRef simulation = new agxSDK::Simulation();
  simulation->stepTo( 8 );
}

class MyThread : public OpenThreads::Thread
{
  public:
    virtual void run()
    {
      // Register this thread as an AgX compute thread.
      // We may after this operation use the AGX API.
      agx::Thread::registerAsAgxThread();

      doStuffInParallel();

      // Unregister when this thread is done. No longer
      // safe to call the AGX API after this operation.
      agx::Thread::unregisterAsAgxThread();
    }
};

int main( int /*argc*/, char** /*argv*/ )
{
  try {
    // Environment setup excluded but should be here.

    // Initialize AGX.
    agx::AutoInit autoInit;

    // This thread is the main thread.
    const agx::Thread* currentThread = agx::Thread::getCurrentThread();
    const agx::Thread* mainThread    = agx::Thread::getMainThread();
    agxAssert( currentThread == mainThread );

    MyThread threads[ 3 ];
    for ( size_t i = 0; i < 3; ++i )
      threads[ i ].run();

    // Wait for threads to finish.
    for ( size_t i = 0; i < 3; ++i )
      threads[ i ].join();
  }
  catch ( const std::exception& e ) {
    std::cerr << "Caught exception: " << e.what() << std::endl;
    return 1;
  }

  return 0;
}
```

agxViewer

- agxViewer

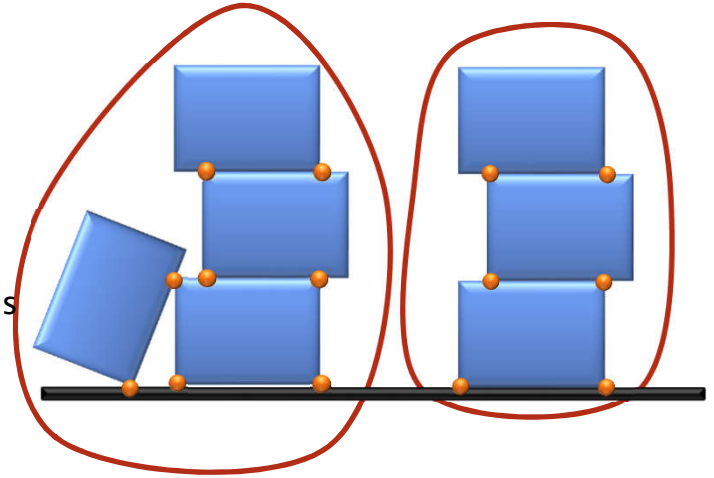
```
> agxViewer beam.agxLua --timeStep 0.01 --startPaused
```

  - Uses OpenSceneGraph for rendering.
  - Can read various files, as well as Python and Lua scripts.
    - .agx, .aagx, agxPy, agxLua, .agxScene (deprecated)
  - Uses agxOSG::ExampleApplication (just provided as example).
  - Most commonly used keys:
    - e – pause/continue simulation
    - r – single step simulation
    - 1 – load scene 1 (reload the file)
    - g – toggle debug rendering
    - G – toggle OSG rendering
    - O – output scene to saved_scene.agx
    - I – read content from saved_scene.agx
    - D – clear scene
    - <Esc> - exit application

Parallelization

- AGX is built with parallelization in mind.
- Tell AGX how many threads should be available:
  - agx::setNumThreads(UInt)
  - 0 – number of threads is set to the number of available cores.
- Various parallelized stages.
  - Narrow phase collision detection.
  - Velocity/position integration.
  - Solver.
- The *partitioner* can split a larger system into sub-systems (dependent on interaction between parts).

Blue – dynamic bodies
Black – static bodies
Orange – contacts



Coupling to external simulation

- In general a very hard problem to get co-simulation numerically stable.
  - Works for loosely coupled systems: Hydrodynamic forces of slowly moving objects.

- AGX
  - Input:
    - Mass, Inertia, Added mass
    - Force/torque
  - Output
    - Linear/Angular velocity
    - (Acceleration)
    - Transformation

- External
  - Output
    - Mass, Inertia, Added mass
    - Force/torque
  - Input
    - Linear/angular velocity
    - Acceleration
    - Transformation

Stable simulations

What has to be regarded in order to achieve stable simulations?

- Mass ratio, damping
- Time step, size, velocities
- Valid initial states

Mass ratio, Damping

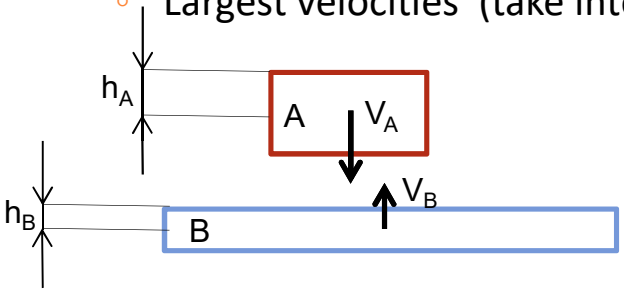
- Mass ratio:
  - Large mass ratios can lead to undesirable simulations when iterative solvers are used.
  - The direct solver in AGX can in general handle very large mass ratios.
  - Practical example
    - Oilrig – Shackle - Wire
      - 100 kg / 18000E3 kg -> 1:180000
  - Unpractical example
    - Attaching a 1E9 kg object to a 10 µm radius wire (~5E-5 kg in total).

- Damping
  - By default bodies are not damped in AGX!
  - In reality we have transformations of energy that are not supported by AGX.
    - Collision -> sound, friction -> heat, etc.
- Velocity damping
  - Linear
  - Angular
  - Body coordinates

```
// Assigns linear velocity damping to all directions (x, y, z) of rb.
rb->setLinearVelocityDamping( 0.2 );
// Assigns angular velocity damping to all directions (x, y, z) of rb.
rb->setAngularVelocityDamping( 0.6 );
```

Time step, size, velocities

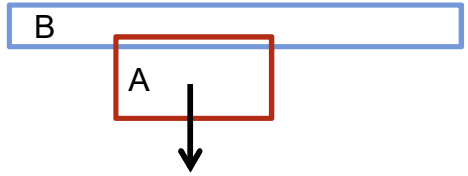
- dt – a discrete time step. Preferably chosen constant.
- Small time steps
  - Slow
- Too large:
  - Fast!
  - Large interpenetrations -> large stabilization forces
  - Missing overlaps (“tunneling”)
  - Large constraint errors
  - Incorrect/Unstable simulation
- How to choose a time step?
  - Largest velocities (take into consideration angular velocity!) vs. smallest dimensions



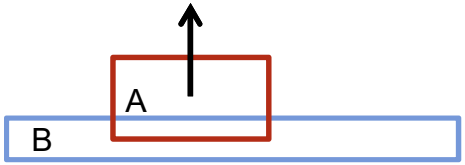
$$V_{rel} = V_A - V_B$$

$$S_{dt} = V_{rel} \cdot dt$$

$$d_t : [S_{dt} << \min(h_a, h_b)]$$



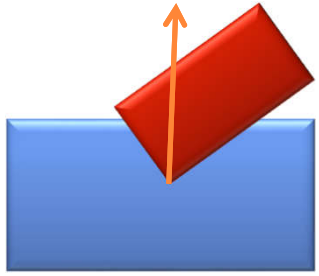
dt too large



dt small enough.

Valid initial states

- The underlying model is only valid for correct initial states.
- Examples of invalid initial states:
  - *Deep overlaps between geometries*
    - Results in large forces -> high velocities
  - *Unrealistic forces (on motors etc.)*
  - *Unrealistic velocities*
  - *Large violation on constraints*
- The result can be unstable simulations.



Section 2

API USAGE

Math classes

- Floating point numbers
  - agx::Real – usually double.
- 2, 3 and 4D vectors
  - agx::Vec2, agx::Vec3, agx::Vec4
  - Operators: +, -, *, ^ (cross product) etc.
- agx::AffineMatrix4x4
  - No scaling
  - Invert, transform vectors and/or points etc.
  - Operators: *, (), <, ==, != etc.
- Rotations
  - agx::Quat
  - agx::Matrix3x3
  - agx::EulerAngles
- Conversion routines between most representations
  - agx::Quat ⟷ agx::AffineMatrix4x4 ⟷ agx::EulerAngles
- Right handed coordinate system

```
agx::AffineMatrix4x4 matrix;
// Set translational part of the matrix.
matrix.setTranslate( 1, 2, 3 );
// Set rotation given Euler angles.
matrix.setRotate( agx::EulerAngles( 0, agx::PI / 2, agx::PI_4 ) );
// Set rotation which will transform a vector from
// x-axis to z-axis.
matrix.setTranslate( 0, 0, 0 );
matrix.setRotate( agx::Quat::rotate( agx::Vec3::X_AXIS(), agx::Vec3::Z_AXIS() ) );
```

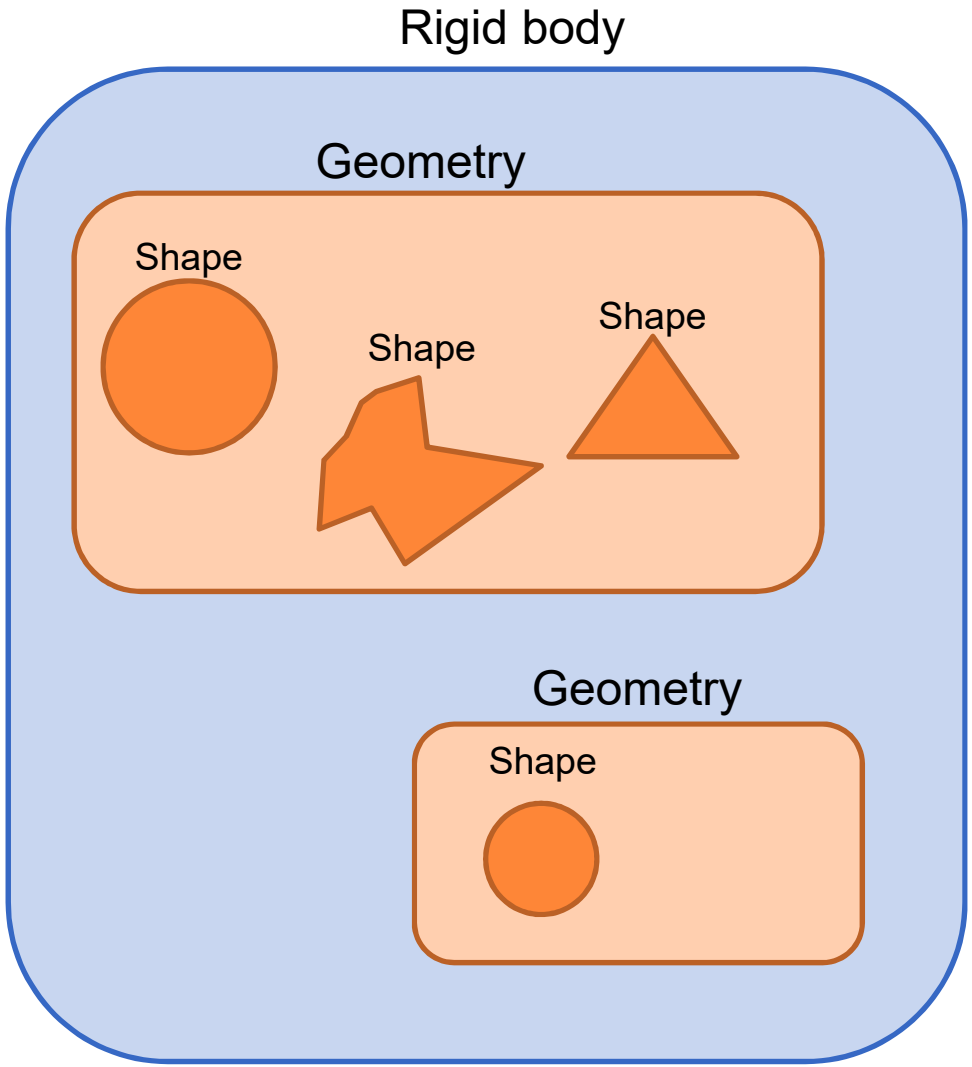
agx::Frame

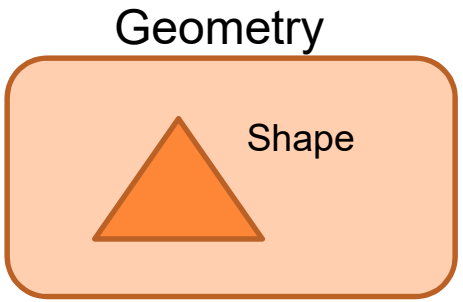
- Defines a coordinate system.
- Can be chained (parent/child).
- Commonly used for modelling constraints.
- agx::RigidBody carries two:
  - Model frame
  - Center of mass/gravity frame
- setTranslate vs. setLocalTranslate
  - setLocal* changes relative to the parent frame.
- Contains methods for transforming points and vectors.

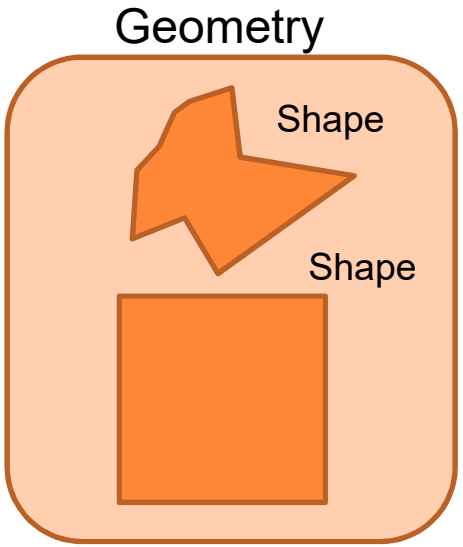
Common methods

```
get/setMatrix( agx::AffineMatrix4x4 matrix )
get/setLocalMatrix( agx::AffineMatrix4x4 localMatrix )
get/setTranslate( agx::Vec3 translate )
get/setLocalTranslate( agx::Vec3 localTranslate )
get/setRotate( agx::Quat rotate )
get/setLocalRotate( agx::Quat localRotate )
transformPointToWorld( agx::Vec3 localPoint )
transformVectorToWorld( agx::Vec3 localVector )
transformPointToLocal( agx::Vec3 worldPoint )
transformVectorToLocal( agx::Vec3 worldVector )
```

Structure


Rigid body
Geometry
Shape
Shape
Shape
Geometry
Shape

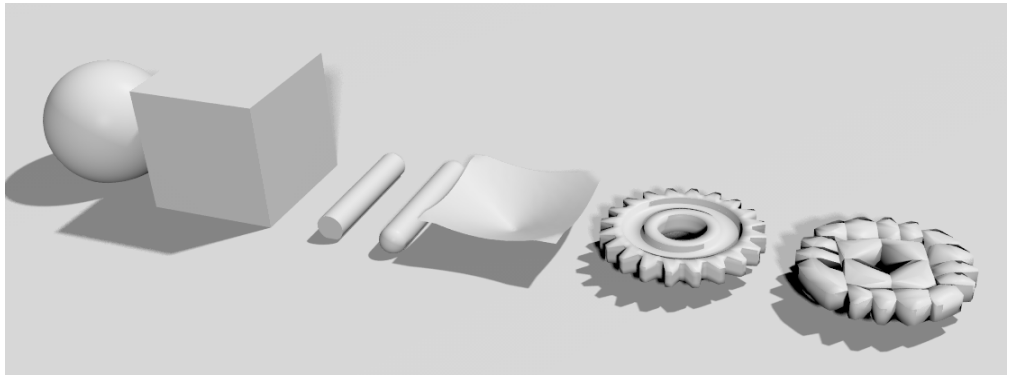

Geometry
Shape


Geometry
Shape
Shape

agxCollide::Shape

- Base class for all geometrical representations
- Native types
  - agxCollide::Box – half extents (x, y, z)
  - agxCollide:: Sphere – radius
  - agxCollide:: Capsule – radius, length
  - agxCollide:: Cylinder – radius, length
  - agxCollide:: Plane – point, normal
  - agxCollide:: Line – p1, p2
  - agxCollide::Mesh – base class for mesh shape types
    - agxCollide::Heightfield
    - agxCollide::Convex
    - agxCollide::Trimesh
    - agxModel::Terrain

```
agx::Real radius = 0.5;
agx::Real mass = 10.0;
agxCollide::SphereRef sphere = new agxCollide::Sphere( radius );
// All shapes knows its volume.
agx::Real volume = sphere->getVolume();
// Local inertia tensor for the shape - input mass.
agx::SPDMatrix3x3 inertia = sphere->calculateInertia( mass );
```



Collider Matrix

- Collider – an algorithm to determine the geometric overlap between two shapes.

	Box	Capsule	Convex	Cylinder	Line	Plane	Sphere	Trimesh	Composite	HeightField
Box	X	X	X	X	X	X	X	X	X	X
Capsule	X	X	X	X	X	X	X	X	X	X
Convex	X	X	X	X	X	X	X	X	X	X
Cylinder	X	X	X	X	X	X	X	X	X	X
Line	X	X	X	X	X	X	X	X	X	X
Plane	X	X	X	X	X	-	X	X	X	-
Sphere	X	X	X	X	X	X	X	X	X	X
Trimesh	X	X	X	X	X	X	X	X	X	X
Composite	X	X	X	X	X	X	X	X	X	X
HeightField	X	X	X	X	X	-	X	X	X	-

Available primitive tests. × Full functionality, (×) Reduced functionality, — Not needed (since both are supposed to be static).

agxCollide::Geometry

- Placeholder for shapes.
- Belongs to agxCollide::Space.
- Holds an agx::Material.
- Can belong to an agx::RigidBody.
- Can be a sensor.
  - Generate contact points but no contact constraints (solver won't see them).

```
geometry->setSensor( true );
```

- Can have surface velocity (basic conveyor)

```
// The surface of this geometry has a velocity of 1 m/s
// in the local x-direction of the geometry.
geometry->setSurfaceVelocity( agx::Vec3f( 1, 0, 0 ) );
```

- API for enabling/disabling collision

```
// 1. Global disable for collisions.
geometry->setEnableCollisions( false );
// 2. Disable collisions against another geometry.
geometry->setEnableCollisions( otherGeometry, false );
// 3. Group id's.
geometry->addGroup( 1 );
otherGeometry->addGroup( 2 );
space->setEnablePair( 1, 2, false );
// 4. Group names.
geometry->addGroup( "one" );
otherGeometry->addGroup( "two" );
space->setEnablePair( "one", "two", false );
```

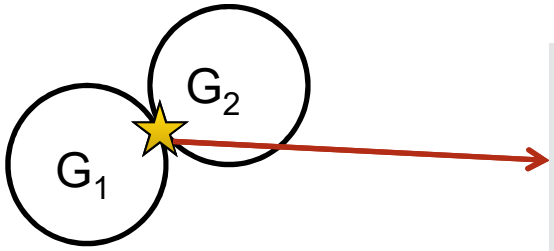
Common methods

```
get/setEnable( bool enable )
get/setPosition( agx::Vec3 position )
get/setTransform( agx::AffineMatrix4x4 transform )
setEnableCollisions( agxCollide::Geometry* other, bool enable )
get/setSurfaceVelocity( agx::Vec3 velocity )
get/setMaterial( agx::Material* material )
setSensor( bool sensor )
isSensor()
addGroup( UInt32 id )
add( agxCollide::Shape* shape, agx::AffineMatrix4x4 relTransform )
```

agx::Material

- agx::SurfaceMaterial
  - Roughness
- agx::BulkMaterial
  - Viscosity (1-restitution)
  - Density
  - Young's modulus
- agx::WireMaterial
  - Friction coefficients (dependent on contact model)
    - Friction coefficient along the wire
    - "Kinematic node velocity scale"
  - Young's modulus stretch
  - Young's modulus bend
- At contact, the resulting attributes are calculated

```
// Two materials; plastic and steel.
agx::MaterialRef plastic = new agx::Material( "Plastic" );
agx::MaterialRef steel   = new agx::Material( "Steel" );

// Surface roughness of the materials.
plastic->getSurfaceMaterial()->setRoughness( 0.05 );
steel->getSurfaceMaterial()->setRoughness( 0.15 );

// Bulk stiffness.
plastic->getBulkMaterial()->setYoungsModulus( 40.0E9 ); // 40 GPa.
steel->getBulkMaterial()->setYoungsModulus( 210.0E9 ); // 210 GPa.

// Bulk density, affects the mass of the object and e.g,
// aerodynamics and hydrodynamics.
plastic->getBulkMaterial()->setDensity( 500.0 ); // kg/m^3
steel->getBulkMaterial()->setDensity( 7750.0 ); // kg/m^3

// Assign materials to the geometries.
geometry1->setMaterial( plastic );
geometry2->setMaterial( steel );
```



$$friction = \sqrt{roughness_1 \times roughness_2}$$

$$restitution = \sqrt{(1 - viscosity_1) \times (1 - viscosity_2)}$$

$$youngsModulus = \frac{Y_1 \times Y_2}{Y_1 + Y_2}$$

agx::ContactMaterial

- Implicit agx::ContactMaterial
  - When two agx::Material interact.
  - Crude approximation of reality.
  - Calculated based on the equations on slide earlier.
- Explicit agx::ContactMaterial – *"material table"*
  - Create an explicit table of material properties.
  - Override all values specified in the agx::Material.

Common methods

```
get/setRestitution( agx::Real restitution )
get/setFrictionCoefficient( agx::Real coeff,
FrictionDirection dir )
get/setYoungsModulus( agx::Real youngs )
```

Friction	Plastic	Steel	Wood
Plastic	0.7	0.5	0.6
Steel	-	0.1	0.2
Wood	-	-	0.3

```
// Three materials; plastic, steel and wood.
agx::MaterialRef plastic = new agx::Material( "Plastic" );
agx::MaterialRef steel   = new agx::Material( "Steel" );
agx::MaterialRef wood    = new agx::Material( "Wood  " );

agxSDK::MaterialManagerRef mgr = simulation->getMaterialManager();

// Explicit contact materials.
agx::ContactMaterialRef plasticPlastic = mgr->getOrCreateContactMaterial( plastic, plastic );
agx::ContactMaterialRef plasticSteel = mgr->getOrCreateContactMaterial( plastic, steel );
agx::ContactMaterialRef plasticWood = mgr->getOrCreateContactMaterial( plastic, wood );
agx::ContactMaterialRef steelSteel = mgr->getOrCreateContactMaterial( steel, steel );
agx::ContactMaterialRef steelWood = mgr->getOrCreateContactMaterial( steel, wood );
agx::ContactMaterialRef woodWood = mgr->getOrCreateContactMaterial( wood, wood );

plasticPlastic->setFrictionCoefficient( 0.7 );
plasticSteel->setFrictionCoefficient( 0.5 );
plasticWood->setFrictionCoefficient( 0.6 );
steelSteel->setFrictionCoefficient( 0.1 );
steelWood->setFrictionCoefficient( 0.2 );
woodWood->setFrictionCoefficient( 0.3 );
```

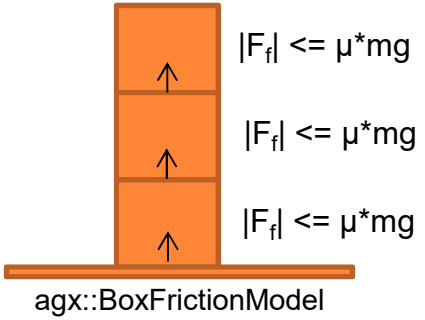
Friction Model

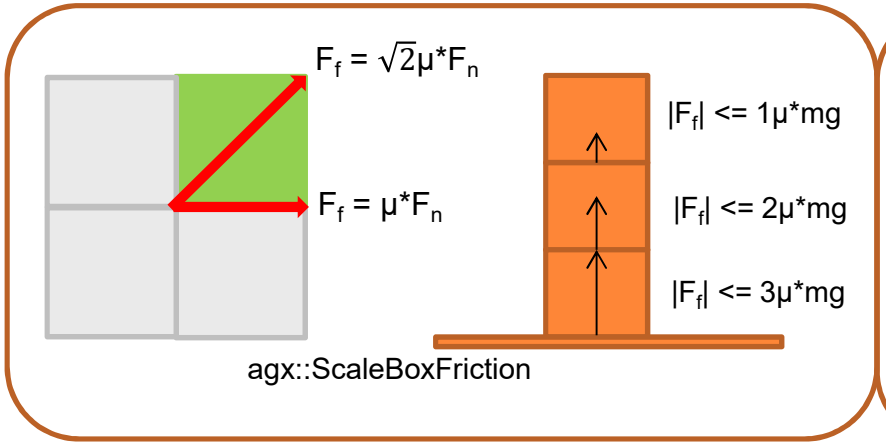
- AGX supports different *friction models and solve algorithms.*
  - agx::FrictionModel and agx::FrictionModel::SolveType.
- Friction model solve types:
  - **agx::FrictionModel::DIRECT**
    - Normal- and friction equations are solved with the high precision direct solver.
  - **agx::FrictionModel::ITERATIVE**
    - Normal- and friction equations are solved with the high speed iterative solver.
  - **agx::FrictionModel::DIRECT_AND_ITERATIVE**
    - Normals are solved in the direct solver. Friction bounds are determined by the iterative solver which becomes input to one last, direct solve.
  - **agx::FrictionModel::SPLIT**
    - Normal forces are calculated by the direct solver and then used in the iterative solver to calculate the friction forces.

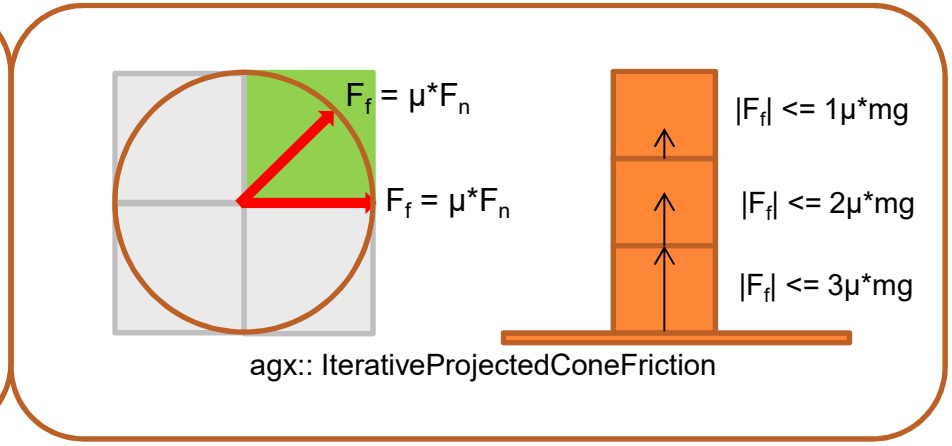
```
// Create a friction model.
agx::FrictionModelRef frictionModel = new agx::IterativeProjectedConeFriction();
// Solve normals and tangents with the iterative solver.
frictionModel->setSolveType( agx::SolveModel::ITERATIVE );
// Assign the friction model to our explicit contact material.
plasticPlastic->setFrictionModel( frictionModel );
```

Friction Model

- Friction model types
  - agx::BoxFrictionModel
    - Static bounds (given by user or roughly estimated).
    - Friction force completely decoupled from the normal force.
  - agx::ScaleBoxFrictionModel (non-linear)
    - Friction force box scaled given current normal force.
  - agx::IterativeProjectedConeFriction (non-linear, default)
    - Same as agx::ScaleBoxFrictionModel, but
    - Recovers the friction cone.







agx::RigidBody

Common methods

```
get/setEnable( bool enable )
get/setMotionControl( MotionControl motionControl )
get/setPosition( agx::Vec3 position )
get/setTransform( agx::AffineMatrix4x4 transform )
add/remove( agxCollide::Geometry* geometry )
getMassProperties()
addForce( agx::Vec3 force )
addTorque( agx::Vec3 torque )
get/setVelocity( agx::Vec3 velocity )
get/setAngularVelocity( agx::Vec3 angularVelocity )
```

- Holds geometries and mass properties.
- Can have velocity and be constrained.
- Motion controls:
  - DYNAMICS – fully dynamic, can be affected by forces.
  - STATIC – velocity always zero, will not move, infinite mass.
  - KINEMATICS – can be controlled with setVelocity/moveTo.
- Has two coordinate systems:
  - Model – setPosition, setTransform, ...
  - Center of mass – setCmPosition, setCmTransform etc...
  - The relative transform between model- and CM frame is constant as long as the mass distribution isn't changed.
- All velocities are specified/returned in world coordinate system.
- Automatically calculates mass properties when:
  - adding/removing a geometry.
  - adding/removing a shape from an associated geometry.
  - assigning new material.
- *NOT automatically when parameters in a material are changed!*

agx::MassProperties

- Stores mass attributes for the RigidBody:
  - Mass
  - Inertia
  - *Added mass coefficients*
    - For marine applications, where the ship hull carries water.
    - Does not affect the gravitational mass.

Common methods

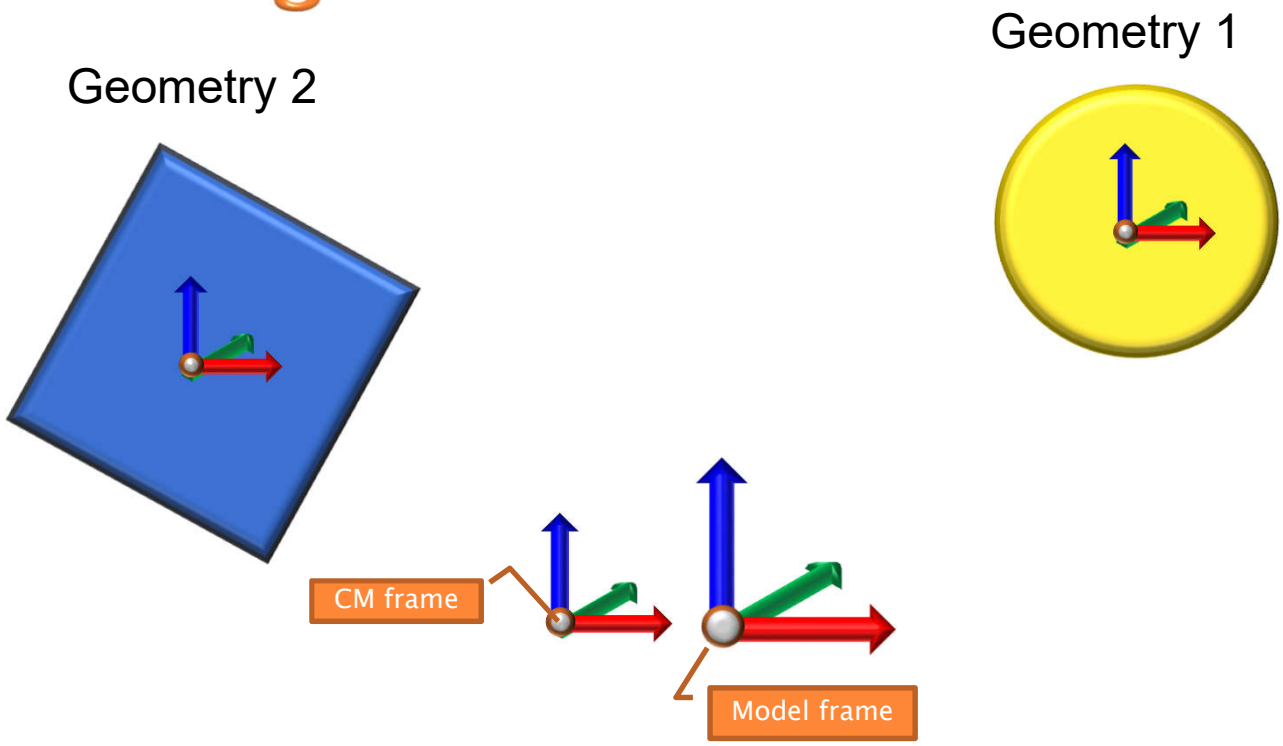
```
get/setMass( agx::Real mass )
setInertiaTensor( agx::SPDMatrix3x3 tensor )
setInertiaTensor( agx::Vec3 diagonal )
get/setMassCoefficients( agx::Vec3 diagonal )
```

```
// Effective mass along local x to be 1.2 * getMass().
rb->getMassProperties()->setMassCoefficients( agx::Vec3( 0.2, 0, 0 ) );
```

```
agxModel::HydrodynamicsParameters* hydroParams = windAndWaterController->getOrCreateHydrodynamicsParameters( shipShape );
// Calculates or loads storage with data of the flow.
hydroParams->initializeAddedMassStorage( "shipAddedMassData.dat" );
```

Added mass

Modelling frames

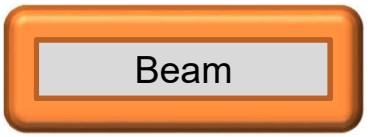


Model frame and CM frame can differ:

- Model frame remains constant
- CM frame gets auto-updated when adding/removing geometries to represent center of mass (default behavior)

Constraints

- Define geometrical relationships between bodies.
- Reduces one or more Degrees Of Freedom (DOF)
- Ordinary constraints:
  - Hinge, Prismatic, LockJoint, DistanceJoint, ...
- Secondary constraints (on one DOF)
  - Controls constraint angles (distance or rotation).
  - TargetSpeedController (Motor1D), LockController (Lock1D), RangeController (Range1D), ScrewController (Screw1D).
- Each DOF can be “relaxed” using compliance.
- Constraints can return the force/torque applied to each body:
  - Use Constraint::setEnableComputeForces(bool) to enable this functionality (once)
    Every time step, use
    - *agx::Bool getLastForce(const agx::RigidBody* rb, agx::Vec3& retForce, agx::Vec3& retTorque, agx::Bool giveForceAtCm = false) const;*
      (or similar alternatives)

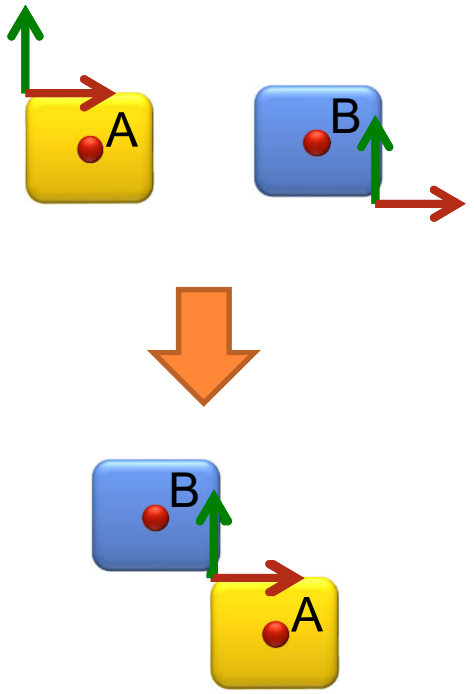


Constraint attachments

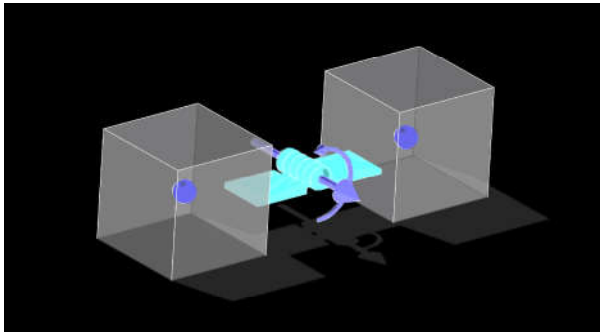
- A frame of reference, relative to an agx::RigidBody.
- When two bodies are constrained, these two frames will in general coincide (but can also be chosen separately for special purposes).
- The z-axes of the constraint attachments are by definition the constraint axes (e.g., hinge and prismatic).

```
// Create an attachment frame relative body A's model origin.
agx::FrameRef frameA = new agx::Frame();
frameA->setLocalTranslate( -0.5, 0, 0.5 );
rbA->addAttachment( frameA, "UpperLeftEdge" )

// Create an attachment frame relative body B's model origin.
agx::FrameRef frameB = new agx::Frame();
frameA->setLocalTranslate( 0.5, 0, -0.5 );
rbB->addAttachment( frameB, "LowerRightEdge" )

// Create a prismatic using the attachments
agx::PrismaticRef lock = new agx::prismatic( rbA, rbA->getAttachment( "UpperLeftEdge" ),
                                             rbB, rbB->getAttachment( "LowerRightEdge" ) );
```

Attachments



agx::Hinge

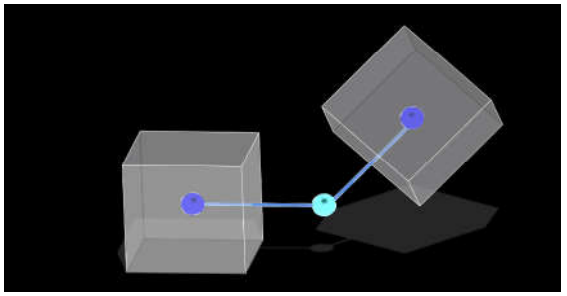


- Removes 5 DOF, leaves one rotational DOF.
- Usage
  - Rotating objects: wheels, drums, doors etc.
- Secondary constraints
  - agx::TargetSpeedController (Motor1D)
    - Attributes: enable, speed (rad/s), force range (Nm).
  - agx::LockController (Lock1D)
    - Attributes: enable, position (rad), force range (Nm).
  - agx::RangeController (Range1D)
    - Attributes: enable, range (rad), max range- and adhesive torque (Nm).
  - You can ask each constraint row which force/torque it applied:
    - getCurrentForce(rowNumber – use each constraint's DOF-enum), or e.g.,
    - constraint->getMotor1D()->getCurrentForce()

Hinge

```
// Constraint frame relative rb1.
agx::FrameRef frame1 = new agx::Frame();
// Constraint frame relative rb2.
agx::FrameRef frame2 = new agx::Frame();

// Create hinge given the two rigid bodies and their frames.
agx::HingeRef hinge = new agx::Hinge( rb1, frame1, rb2, frame2 );
// Enable the motor (default disabled) and set speed.
hinge->getMotor1D()->setEnable( true );
hinge->getMotor1D()->setSpeed( 1 );
// To read last torque applied by the motor.
agx::Real lastTorque = hinge->getMotor1D()->getCurrentForce();
```

agx::BallJoint

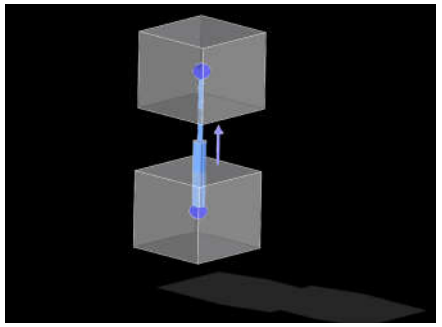


- Removes 3 DOF (translation), leaves 3 rotational DOF
- No secondary constraints.

Ball joint

```
// Constraint frame relative rb1.
agx::FrameRef frame1 = new agx::Frame();
// Constraint frame relative rb2.
agx::FrameRef frame2 = new agx::Frame();

// Position the frames.
frame1->setLocalTranslate( 0, 0, -1 );
frame2->setLocalTranslate( 0, 0,  1 );

agx::BallJointRef ballJoint = new agx::BallJoint( rb1, frame1, rb2, frame2 );
```

agx::Prismatic



- Removes 5 DOF, leaves one translational DOF.
- Usage: suspension, hydraulic actuator etc.
- Secondary constraints
  - agx::TargetSpeedController (Motor1D)
    - Attributes: enable, speed (m/s), force range (N).
  - agx::LockController (Lock1D)
    - Attributes: enable, position (m), force range (N).
  - agx::RangeController (Range1D)
    - Attributes: enable, range (m), max range- and adhesive force (N).

Prismatic

```
// Create the prismatic given local offset/rotation relative body 1.
// Attached in world, so 'rb2' is null.
agx::Vec3 relPos = agx::Vec3( 0, 0, 1 );
agx::Vec3 axis   = agx::Vec3::Y_AXIS();
agx::PrismaticRef prismatic = agx::Constraint::createFromBody< agx::Prismatic >( relPos, axis, rb1,
nullptr );
// Enable the range (default disabled).
prismatic->getRange1D()->setEnable( true );
prismatic->getRange1D()->setRange( -1, 1 );
```

agx::LockJoint

- Removes 6 DOF, leaving none.
- No secondary constraints.
- Usage: Locking bodies together.

Lock joint

```
// Create given position and axis in world coordinates.
agx::Vec3 worldPos = agx::Vec3( 2, 0, 2 );
agx::Vec3 worldAxis = agx::Vec3( 0, 1, 0 );
agx::LockJointRef lock = agx::Constraint::createFromWorld< agx::LockJoint >( worldPos, worldAxis, rb1, nullptr );
// Relax the lock in all six DOFs.
lock->setCompliance( 3E-9 );
```

agx::DistanceJoint

- Controls 1 translational DOF.
- Secondary constraints
  - Same as prismatic; Lock1D, Motor1D and Range1D.
  - Default Lock1D enabled.
- Usage: Optimized hydraulic cylinder setup.

Distance joint

```
// Constraint frame relative rb1.
agx::FrameRef frame1 = new agx::Frame();
// Constraint frame relative rb2.
agx::FrameRef frame2 = new agx::Frame();

// Position the frames.
frame1->setLocalTranslate( 0, 0, -2 );
frame2->setLocalTranslate( 0, 0,  2 );

agx::DistanceJointRef distanceJoint = new agx::DistanceJoint( rb1, frame1, rb2, frame2 );
```

Other constraints

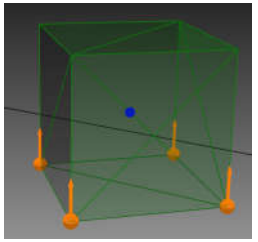
- agx::PlaneJoint
  - Point in plane.
- agx::Dot1Joint
  - Constraint axes orthogonal.
- agx::AngularLockJoint
  - Locked relative orientation.
- agx::CylindricalJoint
  - Prismatic and hinge, i.e., motion along and about the constraint axis.
- agx::UniversalJoint
  - Cardan, transmits rotary motion.
- agx::PrismaticUniversalJoint
  - Universal joint but with additional motion along the axis.

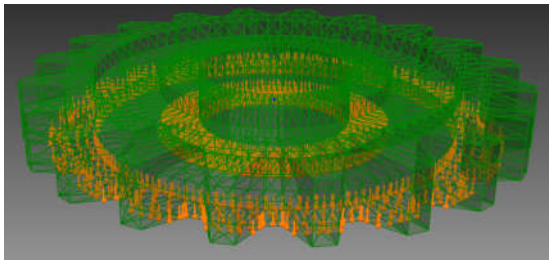
agxCollide::GeometryContact

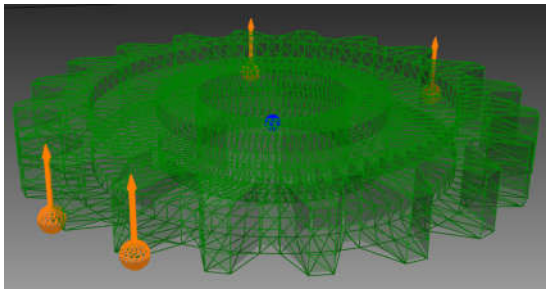
- Contains contact information calculated by agxCollide::Space.
- agxCollide::GeometryContact
  - agxCollide::Geometry* geometry(size_t ith)
    - First or second geometry.
  - agx::RigidBody* rigidBody(size_t ith)
    - First or second rigid body (if any, may be null).
  - agxCollide::ContactPointVector& points()
    - Set of contact points.
- agxCollide::ContactPoint
  - agx::Vec3 point() // Point in world.
  - agx::Vec3f normal() // Contact normal direction.
  - agx::Real depth() // Contact penetration depth.
  - agx::Vec3 getForce() // Contact force (after solve!).
  - agx::Real magnitude(size_t) // Magnitude of normal- and tangential forces.
  - Bool enabled() // Is the point enabled? Contact reduction (next slide).

Contact Reduction

- Consider a resting box on a plane
  - 4 contact points. Ok.
- Consider a general triangle mesh
  - 1000:s of contact points
  - Overdetermined; takes more time to solve.
  - With contact reduction: ~4 contact points.
- Finds the largest spanning area.
- Contact reduction can also be used between RigidBody/RigidBody contacts.
- For higher fidelity, a bin resolution of 3 should be chosen (2 is default, for higher computational speed)



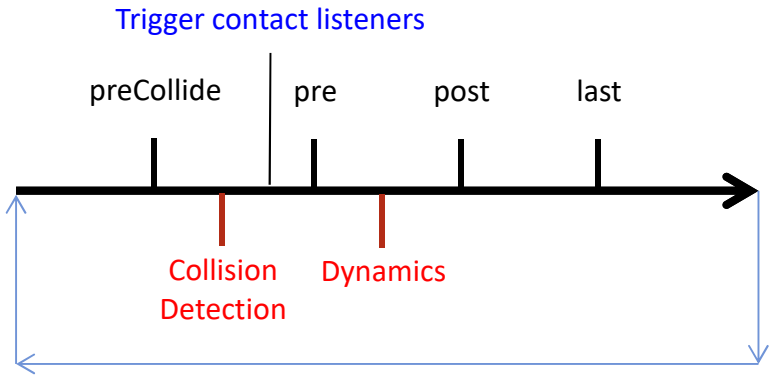




```
// Disable contact reduction.
plasticPlastic->setContactReductionMode( agx::ContactMaterial::REDUCE_NONE );
// Default: Per geometry pair.
plasticPlastic->setContactReductionMode( agx::ContactMaterial::REDUCE_GEOMETRY );
// Contact reduction per body pair (several geometries each).
plasticPlastic->setContactReductionMode( agx::ContactMaterial::REDUCE_ALL );
// Contact reduction bin resolution for whole space.
sim->getSpace()->setContactBinResolution( 3 );
// Contact reduction bin resolution per contact material (overriding space).
plasticPlastic->setContactBinResolution( 3 );
```

Listeners

- agxSDK::EventListener – base class
  - Enables user code in the simulation loop.
  - Implement virtual methods.
  - Add to a Simulation.



agxSDK::StepEventListener

- agxSDK::StepEventListener
  - preCollide – before collision detection.
  - pre – *after* collision detection, *before* dynamics.
  - post – *after* dynamics (new positions, velocities, forces).
  - last –last in the simulation loop (before time update).
  - lastLastLastLast – not supported.
    - However, a priority can be set per StepEventListener.

```
class MyStepEventListener : public agxSDK::StepEventListener
{
  public:
    MyStepEventListener()
    {
      setMask( PRE_COLLIDE | PRE_STEP | POST_STEP | LAST_STEP );
    }

    virtual void preCollide( const agx::TimeStamp& ) AGX_OVERRIDE {}
    virtual void pre( const agx::TimeStamp& ) AGX_OVERRIDE {}
    virtual void post( const agx::TimeStamp& ) AGX_OVERRIDE {}
    virtual void last( const agx::TimeStamp& ) AGX_OVERRIDE {}

  protected:
    virtual ~MyStepEventListener() {}
};

simulation->add( new MyStepEventListener() );
```

agxSDK::ContactEventListener

- Listen to contact events.
  - impact – first contact between two geometries.
  - contact – continuous contact state (between impact and separation).
  - separation – at separation of the two geometries.
- agxSDK::ExecutionFilter
  - Implements a matching algorithm.
  - Filters out agxCollide::GeometryContact/agxCollide::GeometryPair.
  - e.g. agxSDK::GeometryFilter (there are many other ones)
    - Filter out contacts given one or two geometries (i.e., any object colliding with this geometry and if this and the other geometry collides).

```
class MyContactEventListener : public agxSDK::ContactEventListener
{
  public:
    MyContactEventListener() { setMask( IMPACT | CONTACT | SEPARATION ); }
    virtual KeepContactPolicy impact( const agx::TimeStamp& t, agxCollide::GeometryContact* gc ) AGX_OVERRIDE
    {
      // Contact kept if impacting (i.e., the geometries didn't overlap last update).
      return KEEP_CONTACT;
    }
    virtual KeepContactPolicy contact(const agx::TimeStamp& t, agxCollide::GeometryContact* gc ) AGX_OVERRI
    {
      // Remove contact, other contact listeners are able to see this contact,
      // the solver wont.
      return REMOVE_CONTACT;
    }
    virtual void separation( const agx::TimeStamp& t, agxCollide::GeometryPair& gp ) AGX_OVERRIDE
    {
      // The geometries don't overlap anymore.
    }
  protected:
    virtual ~MyContactEventListener() {}
};
```

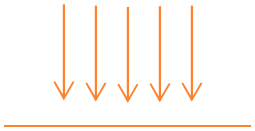
Space

Listeners

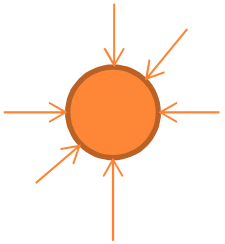
```
preCollide:     0.48333 s
  - impact:     0.48333 s between capsule and ground
pre:            0.48333 s
post:           0.48333 s
last:           0.48333 s

preCollide:     0.50000 s
  - contact:    0.50000 s between capsule and ground
pre:            0.50000 s
post:           0.50000 s
last:           0.50000 s

preCollide:     0.51667 s
  - separation: 0.51667 s between ground and capsule
pre:            0.51667 s
post:           0.51667 s
last:           0.51667 s
```

agx::GravityField

- Applies the gravity force on every enabled agx::RigidBody.
- agx::UniformGravityField



- agx::PointGravityField



agxSDK::Assembly/Collection

- Collection of bodies, constraints, geometries, other assemblies, listeners
- Assembly: Can be transformed.
- Collection, same as Assembly but without the transformation.
- Can be used to assemble whole or parts of a construction.
- When the assembly is transformed, so is its content.

```
agxSDK::AssemblyRef assembly = new agxSDK::Assembly();
assembly->add( rb1 );
assembly->add( rb2 );
assembly->add( rbA );
assembly->add( rbB );

assembly->add( hinge );
assembly->add( prismatic );

assembly->add( new MyStepEventListener() );

// Will transform all objects in this assembly.
assembly->setPosition( newPosition );
assembly->setRotation( newRotation );
```

Assembly

Common methods

```
get/setPosition( agx::Vec3 position )
get/setRotation( agx::Quat rotate )
add/remove( agx::RigidBody* rb )
add/remove( agxCollide::Geometry* geometry )
add/remove( agx::Constraint* constraint )
add/remove( agxSDK::EventListener* listener )
add/remove( agxSDK::Assembly* assembly )
```

agxWire::Wire

- Hybrid, multi-resolution wires with massless frictional contacts.
  M. Servin, C. Lacoursière, K. Bodin, IEEE Transactions on Visualization and Computer Graphics, Volume: 17 Issue:7, On page(s): 970 - 982, July (2011). IEEE computer Society Digital Library. IEEE Computer Society
- agxWire is a *large* API.
- An agxWire::Wire
  - carries a material.
  - Interface for reading tension, add/insert/remove nodes, render aid etc.
  - Can be cut in two and/or merged with other wires.
- Adaptive resolution:
  - Resolution as input – can be changed during simulation.
  - Nodes comes and go during simulation based on tension, contacts etc..
  - A wire can locally have a user defined higher resolution.
- Length of wire is automatically calculated during the *routing process*.
- Generally, no twisting simulated (with exceptions such as agxWire::Link).

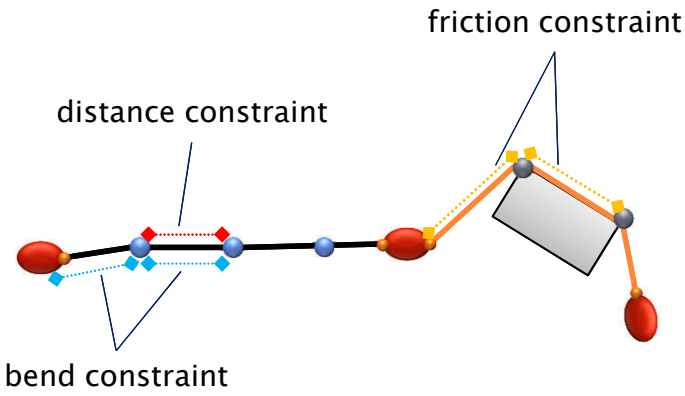

```
agx::Real radius = 0.02;
agx::Real resolution = 2.0; // Two mass elements per meter.
agxWire::WireRef wire = new agxWire::Wire( radius, resolution );
// Routing the wire with three nodes.
wire->add( new agxWire::FreeNode( -2, 0, 0 ) );
wire->add( new agxWire::FreeNode(  0, 0, 3 ) );
wire->add( new agxWire::FreeNode(  2, 0, 0 ) );

simulation->add( wire );
```

agxWire::Wire - material

- The wire is modeled using constraints, with the following parameters:
  - Distance constraints (between two nodes):
    - wire->getMaterial()->getWireMaterial()->setYoungsModulusStretch(agx::Real)
  - Bend constraints (between three nodes):
    - wire->getMaterial()->getWireMaterial()->setYoungsModulusBend(agx::Real)
  - Friction constraint (default contact model):
    - wire->getMaterial()->getWireMaterial()->setFrictionCoefficient(agx::Real)
  - To model e.g. a chain, set Young's modulus bend to a small (~1) value.

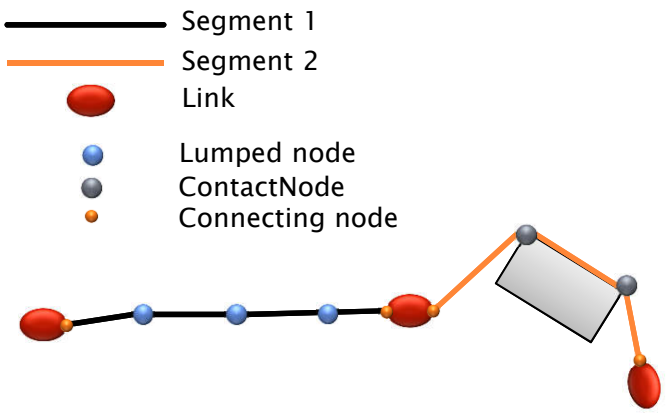
```
// Chain-like behavior if we relax the bend constraints.
wire1->getMaterial()->getWireMaterial()->setYoungsModulusBend( 100 );
// Stiff pipe-like behavior if we increase the bend stiffness.
wire2->getMaterial()->getWireMaterial()->setYoungsModulusBend( 1.0E12 );
// Rubber band.
wire3->getMaterial()->getWireMaterial()->setYoungsModulusStretch( 1.0E8 );
```

Wire material



agxWire::Wire - collisions

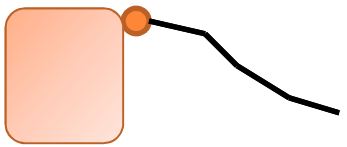
- Default wire contact model:
  - Explicit contact node movements.
  - Handles unrealistically high tension.
  - Handles five shape types:
    - agxCollide::Cylinder, agxCollide::Box, agxCollide::Convex, agxCollide::Trimesh and agxCollide::HeightField
  - Geometries may only contain one (1) shape.



- Dynamic wire contact model:
  - Adds bodies to the wire during contact for the solver to calculate correct interaction forces/velocities.
  - Handles high tension but will start to vibrate for very high loads.
- Wires can collide with each other (under certain circumstances).
  - Disabled by default.
  - `agxWire::WireController::instance()->setEnableCollisions( wire1, wire2, true ).`

agxWire::WinchController

- Constraint-based winch.
- Can be attached to an agx::RigidBody.
- Winch in/out wire.
- Control:
  - Desired speed
  - Maximum force
  - Brake force
- Auto-feed:
  - Calculates how much wire is needed to be pulled out/in to ensure a non-slacking wire after routing.



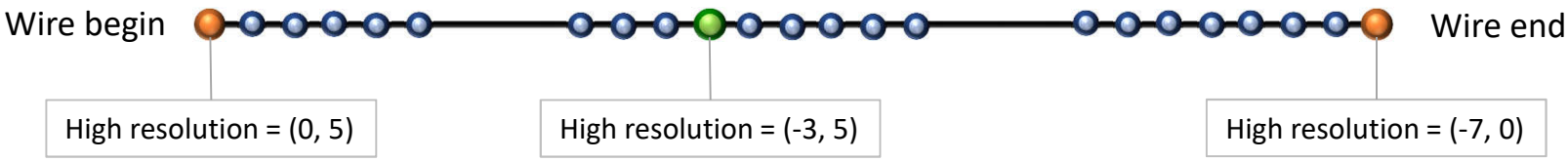
```
// Position of winch given in rb frame.
agx::Vec3 localWinchPosition = agx::Vec3( 0, 0, -0.5 );
// Direction of winch given in rb frame.
agx::Vec3 localWinchDirection = agx::Vec3( 0, 0, -1 );
// How much wire initially on the winch.
agx::Real initialPulledInLength = 50.0; // Meters.
agxWire::WireWinchControllerRef winch = new agxWire::WireWinchController( rb, localWinchPosition, localWinchDirection );

// Route given winch.
wire->add( winch, initialPulledInLength );
wire->add( new agxWire::FreeNode( 0, 0, -5 ) );

// Set speed to pull in wire, 0.4 m/s.
winch->setSpeed( -0.4 );
// Set available maximum force the winch can pull/push with.
winch->setForceRange( 30.0E3 ); // 30 kN
```

agxWire::Node

- A primitive used for *routing* and simulating a wire:
  - agxWire::BodyFixedNode – fixed connection to a body or world.
    - If routed – only valid to be first and/or last node in the wire (i.e., no intermediate fixed nodes).
    - The wire's internal mass nodes have this type.
  - agxWire::ContactNode – default contact models node.
    - Automatically added during interaction with a geometry.
    - May be used to route with as well – takes an agxCollide::Geometry as argument.
  - agxWire::FreeNode – free "temporary" node.
    - If first and/or last node is free, the wire isn't attached at that end.
    - The node is temporary if used during routing.
  - agxWire::EyeNode – node that can slide along the wire.
    - Has two friction coefficients – one forward and the other one backward.
  - agxWire::ConnectingNode – similar to a agxWire::BodyFixedNode but can handle much higher tension.
    - Has an extra node in the center of mass of the object it's attached to.
    - Controls the bend compliance and damping over the node during high tension to maintain stability.

Dynamic resolution

- Stability under tension.
  - If an internal mass node vibrates too much it is removed in order to maintain stability.
  - The node returns when the tension is low enough.
- Resolution can vary over the wire. Controlled via HighResolutionRange:
  - agxWire::HighResolutionRange* range = agxWire::Node::getHighResolutionRange();




```
// Take the high resolution range from the winch node.
agxWire::HighResolutionRange* hr = winch->getStopNode()->getHighResolutionRange();
// The winch is located at the beginning of the wire.
// Let there be high resolution from the winch node
// (0 m), to 5 meters ahead. Default resolution is
// used elsewhere.
agx::Real rangeStart    = 0.0;
agx::Real rangeEnd      = 5.0;
agx::Real newResolution = 6.0;
hr->set( rangeStart, rangeEnd, newResolution );
```

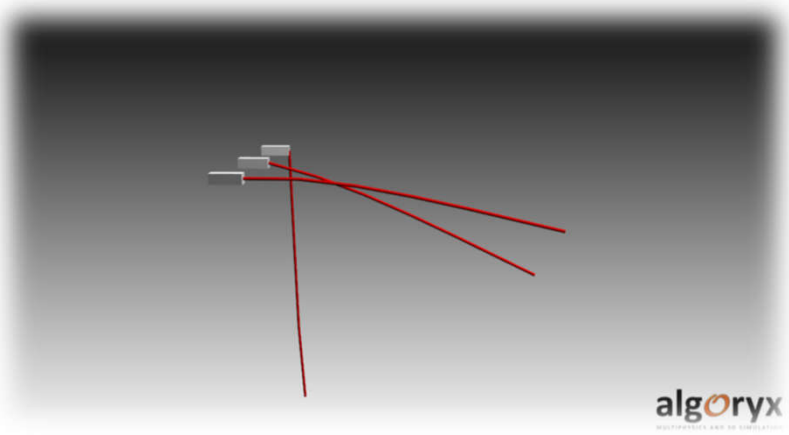
Higher resolution

Wire operations

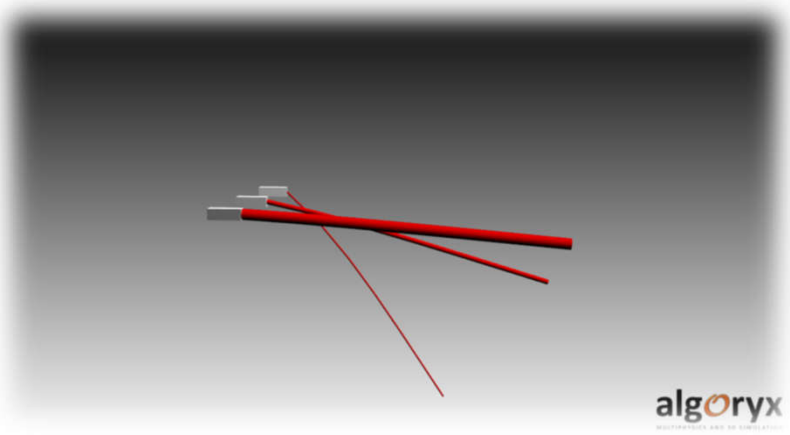
- Wires can be:
  - Cut – into two separate wires.
    - agxWire::WireRef wire2 = wire1->cut(cutPosition);
  - Merged – into one wire.
    - wire1->merge(wire2);
    - Extra wire is added between last node of wire1 to first node of wire2. I.e., the nodes do not have to be close.
  - Reversed – node order reversed.
    - wire->reverse()
    - Often used before merge.
  - Other runtime features:
    - wire->detach(isBegin); // Detach from attachment.
    - wire->attach(newNodeOrWinch, isBegin); // Attach to a winch or a new node.
    - pointOnWire = wire->findPoint(any3DPoint); // Closest point on wire to a given point in 3d space.
    - pointOnWire = wire->findPoint(distanceFromStart);
    - tension = wire->getTension(point);
    - tension = wire->getTension(distanceFromStart);
    - wire->setRadius(newRadius);
    - wire->setResolutionPerUnitLength(newResolution);

agxWire::Link – connection properties

- Connection properties in the connection between a link and a wire.
  - Bend stiffness – how hard it is to bend the wire around the link connection point (default: 0, max: Young's modulus bend of the wire).
  - Twist stiffness – simulates the response of twisting
    - Assumes that the other end has infinite twist resistance,
    - i.e., the object on the other side will not "feel" the link twisting.



Identical radius, different values of bend stiffness.



Identical bend stiffness, different radius.

agxWire::Link – additional features

- agxWire::LinkObjectStabilizationAlgorithm
  - Stabilizing the link object when **in contact** with another object and the connected wires are under high tension (avoiding oscillations.
  - Stern rollers, guide pins etc..

```
// Filter that finds a body the link is in contact with.
agx::String stabilizeTag = "stabilize";
agxWire::LinkObjectStabilizingAlgorithm::RigidBodyPropertyBoolFilterRef filter =
  new agxWire::LinkObjectStabilizingAlgorithm::RigidBodyPropertyBoolFilter( stabilizeTag, link );

// Add the object stabilizing algorithm to the link, given the filter.
link->add( new agxWire::LinkObjectStabilizingAlgorithm( filter ) );

// Fetch an object the link may be in contact with when the connected
// wires are under high tension.
agx::RigidBody* sternRollerBody = ship->getSternRoller();
// Add the property bool tag we passed to filter above.
sternRollerBody->getPropertyContainer()->addPropertyBool( stabilizeTag, true );
```

  - Performance
    - Not all objects need this special treatment; for performance reasons: Apply only where needed.

agxCable::Cable

- Similar (bot not same) as agxWire::Wire
  - 1-dimensional structure: Lumped element approach
  - Routing with agxCable::Node
  - Support bend, stretch *and twisting (which Wire does not)*
  - Fixed resolution (no dynamic resolution like Wire)
  - Full dynamic contact handling and self collision
  - No winch (yet)
  - Elasto-plastic deformation
- Use areas
  - Cables
  - Hoses
  - Pipes

Routing a Cable

1. Create the Cable

```
agx::Real radius = 0.02;
agx::Real resolutionPerLengthUnit = 2;
agxCable::CableRef cable = new agxCable::Cable(radius, resolutionPerLengthUnit);
```

2. Route the Cable (create cable nodes)

```
// Attach (with a LockJoint) Cable relative to a rigidbody
cable->add(new agxCable::BodyFixedNode(rigidBody, agx::Vec3(0, -0.3, -0.5)));

// Point in world coordinates
cable->add( new agxCable::FreeNode( agx::Vec3( 20, -0.3, 0 ) ) );
```

3. Add the Cable to the simulation

```
// Add the cable to a simulation
simulation->add( cable );
```

Routing a Cable

- Note that:
  - When routing a Cable, the segments and the routing points might not match.
  - The routing algorithm at(`simulation->add( cable )`) will try to make the best result.
  - If you want more information of the potential result of the routing, you can call tryInitialize(). It will not create the nodes, just make a ”dry run” and report the result.

```
agxCable::InitializationReport report = cable->tryInitialize();

std::cout << "\nFirst initialization attempt resulted in " << report.getNumNodes() <<
             " nodes and error " << report.getMaximumError() <<
             ". Resolution is " << cable2->getResolution() << std::endl;
```

Routing a Cable

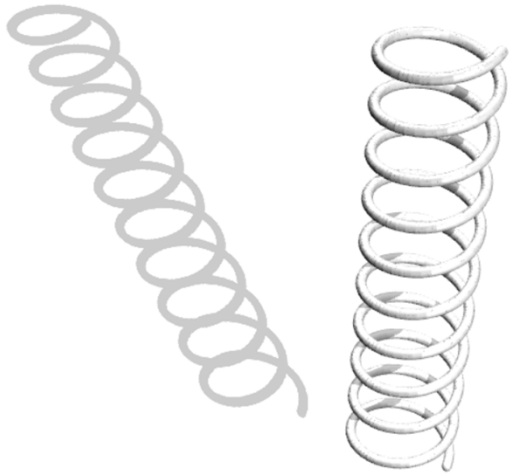
- If you are not happy with the routing result, you can change the maximum allowed error:

```
/* We can say that we accept this error by passing a number slightly
   higher than 0.05 to tryInitialize. This will cause the segmentation algorithm
   to accept the initial resolution dictated by the length of the shortest
   cable leg.
*/
agxCable::InitializationReport report = cable->tryInitialize(agx::Real(0.051));
```

Routing a non-straight Cable

- If you want to create a cable which does not have straight line as an initial rest state, you can do rebind:

```
/*
Route or move the cable in the configuration that you want, for example a spiral,
then call rebind(). Now the cable has a new rest state, allowing you to simulate a
spiral
*/
cable->rebind();
```



Handling contacts with Cable

- If you want to find a Cable belonging to a Geometry, for example in a ContactEventListener:

```
/*
  Using a static helper method in agxCable::Cable we can find the cable, if
  any, that a particular geometry is part of.
*/
agxCable::Cable* cable = agxCable::Cable::getCableForGeometry(geometry1);-
```

Attaching bodies to an existing Cable

- First find the cable segment to which you want to attach a rigid body. You do this with a CableIterator:

```
/*
  In some way, iterate over the segments, and find the one you want to attach to
*/
agxCable::CableIterator rigidSegment = cable->begin();
  while (rigidSegment.getEndPosition().x() < rigidBody->getPosition().x())
    rigidSegment.advance();
```

- Then create an attachment transform OR a point:

```
/*
  Create the transformation matrix. The body transformation relative to the Cable segment transform
*/
agx::AffineMatrix4x4 translate = agx::AffineMatrix4x4::translate(x, y,z);
agx::AffineMatrix4x4 rotate = agx::AffineMatrix4x4::rotate(agx::Z_AXIS, agx::X_AXIS);
agx::AffineMatrix4x4 transform = rotate * translate;

/*
  Attach with a LockJoint (if transform is a Vec3, it will be a BallJoint)
*/
cable->attach(rigidSegment, attachedRigidBody, transform);
```

Accessing attachment properties

- Assume you want to access the constraint of an attached rigid body:

```
// Get an iterator to the segment you want:
 agxCable::CableIterator rigidSegment = cable->begin();

 // Next you access the node
 agx::Constraint *constraint = iterator.getNode()->getAttachment(0)->getConstraint();

 // We will assume it is a lock joint in this example
 agx::LockJoint *lock = dynamic_cast<agx::LockJoint *>(constraint);
```

agxJava

- AGX C++ API export to Java
- JNI (Java Native Interface) using SWIG
- Similar syntax:

C++

```
std::vector< agx::Vec3 > points;
agxWire::RenderIterator it = m_wire->getRenderBeginIterator();
const agxWire::RenderIterator endIt = m_wire->getRenderEndIterator();
while ( !it.equal( endIt ) ) {
  points.push_back( it.get()->getWorldPosition() );
  it.inc();
}
```

Java

```
List< agx.Vec3 > points = new ArrayList< agx.Vec3 >( numNodes );
agxWire.RenderIterator it = m_wire.getRenderBeginIterator();
final agxWire.RenderIterator endIt = m_wire.getRenderEndIterator();
while ( !it.equal( endIt ) ) {
  points.add( it.get().getWorldPosition() );
  it.inc();
}
```

- Easy to extend with custom functionality written in C++, interfaced to Java.

agxJava – custom extensions

MyClass.h

```
#include <agx/RigidBody.h>

class MYLIBEXPORT MyClass
{
  public:
    MyClass( agx::Real massAddition );
    agx::Real increaseMass( agx::RigidBody* body );

  private:
    agx::Real m_massAdd;
};
```

Example_extendingTest.java

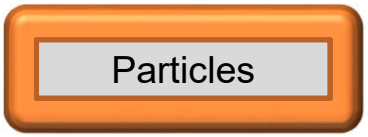
```
public class Example_extendingTest
{
  public static void main(String[] args)
  {
    System.loadLibrary("agxJavaRuntime");
    System.loadLibrary("MyClassCpp");

    agx.agxSWIG.init();
    agx.RigidBody rb = new agx.RigidBody();
    rb.getMassProperties().setMass( 10.0 );
    System.out.printf( "Current mass: %f\n", rb.getMassProperties().getMass() );

    MyPackage.MyClass myClass = new MyPackage.MyClass( 20.0 );
    myClass.increaseMass( rb );

    System.out.printf( "New mass: %f\n", rb.getMassProperties().getMass() );
  }
}
```

MyPackage.i

```
%module MyPackage

%include <windows.i>

%import "../../../configuration/agxJava/agxJava.i"

%typemap(javaimports) SWIGTYPE %{
import agx.*;
%}

%pragma(java) moduleimports=%{
import agx.*;
%}

%pragma(java) jniclassimports=%{
import agx.*;
%}

%{
#include "MyClass/include/export.h"
#include "MyClass/include/MyClass.h"
%}

%import "MyClass/include/export.h"

// And finally we wrap our class.
%include "MyClass/include/MyClass.h"
```

agx::ParticleSystem

- Tutorials/agxOSG/tutorial_particleSystem.cpp.
- No rotation or angular velocity, only position and linear velocity.
- Very efficient for larger systems.
- Can interact with geometries and rigid bodies.
- Hierarchical grid for collision detection.
- Solved with an iterative (GS) solver.
- Can simulate smoke, granular, rocks, etc.



GranularBodySystem

- luaDemos\tutorials\tutorial_granularBodies.agxLua
- Rigid particles with 6DOF (position, rotation). Inherits from agx::ParticleSystem
- Normal forces are computed using Hertzian contact law.
  - Coulomb friction and rolling resistance.
- Can use parallel solver for simulation speedup. (Need to use > 1 threads)
- Also has option to force 32bit float for particle contact solving (slightly faster).

```
// Create and configure the particle system.
agx::Physics::GranularBodySystemRef granularBodySystem = new agx::Physics::GranularBodySystem();
simulation->add(granularBodySystem);
granularBodySystem->setParticleRadius(1.0);
granularBodySystem->setParticleMass(1.0);

// Spawn a bunch of particles in the container.
Bound3 containerBound = Bound3(Vec3(0.0, 0.0, 0), Vec3(5.0, 1.0, 3.0));
Bound3 spawnBound = containerBound * 0.8;
spawnBound.max().x() = containerBound.max().x()*0.4;
spawnBound.max().z() = containerBound.max().z()*1.3;
granularBodySystem->spawnParticlesInBound(spawnBound, Vec3(2.0*granularBodySystem-
>getParticleRadius()), Real(0.3)*granularBodySystem->getParticleRadius() );

// Enable parallel Gauss-Seidel iterative solver (Non-parallel iterative Gauss-Seidel is default)
simulation->getSolver()->setUseParallelPgs(true);

// Enable 32bit granular float solver. Will force solver to use 32 bit float for granulars regardless of agx::Real
definition.
simulation->getSolver()->setUse32bitGranularBodySolver(true);
```

Emitters

- Can use emitters as a source for creating continuous flow of particles.
- Emitter rate depends on assigned quantity: Particle Mass(*kg/s.*), Particle Number(*p/s*) or Particle Volume(m^3/s).
- Emitter will create particles inside the bound of the geometry coupled to the emitter.

```
agx::Physics::GranularBodySystemRef granularBodySystem = new agx::Physics::GranularBodySystem();
simulation->add(granularBodySystem);

ParticleEmitterRef emitter = new agx::ParticleEmitter(granularBodySystem);
agxCollide::GeometryRef sphere = new agxCollide::Geometry(new agxCollide::Sphere(0.6));
emitter->setGeometry(sphere);
emitter->setQuantity(ParticleEmitter::QUANTITY_VOLUME);
emitter->setRate(0.6);
emitter->setGeometry(sphere);
simulation->add(emitter);
```

ParticleDistributionTable

- Can model different particle model distributions.
  - Each model can have a set of Radius, Material and a Weight.
- Can assign distributions to emitters.
- Can specify a weight to each model that governs how much of the total quantity of the distribution should be created for that model.

```
// Create materials
MaterialRef material1 = new Material("material1");
MaterialRef material2 = new Material("material2");
MaterialRef material3 = new Material("material3");

// Create distribution
ParticleDistributionTableRef distribution = new ParticleDistributionTable();
distribution->addModel(new ParticleDistributionModel(0.05, material1, 0.6));
distribution->addModel(new ParticleDistributionModel(0.10, material2, 0.3));
distribution->addModel(new ParticleDistributionModel(0.2, material3, 0.03));

// Couple distribution to emitter
emitter->setDistributionTable(distribution);
```

- Quantities: Particle number, Particle mass, Particle volume.

```
enum ProbabilityQuantity
{
  QUANTITY_COUNT,
  QUANTITY_VOLUME,
  QUANTITY_MASS
};

agx::ParticleDistributionTable(ProbabilityQuantity quantity = QUANTITY_COUNT);
agx::ParticleDistributionModel(agx::Real particleRadius, agx::Material *particleMaterial, agx::Real probabilityWeight);
```

Section 5

Brief overview of the rest of the API

Serialization

- A Simulation can be serialized.
  - Good for debugging customers simulations.
  - Only native AGX data will be written to the serialization.
    - Not customer listeners for example.
  - .agx – binary format for faster read/write
  - .aagx – XML
- agxViewer:
  - ’O’ – write scene to saved_scene.agx
  - ’I’ – read scene from saved_scene.agx
- In code:
  - agxIO::writeFile(<path>, simulationPtr);
  - agxIO::readFile(<path>, simulationPtr);

Serialization

- Store and restore of a simulation:

```
agx::String binaryFormat = ".agx";
agx::String asciiFormat  = ".aagx";
// Save current state to ASCII XML format.
simulation->write( "dump" + asciiFormat );
// Save current state to binary format.
simulation->write( "dump" + binaryFormat );

// Read back the binary file to another simulation.
agxSDK::SimulationRef newSimulation = new
agxSDK::Simulation();
newSimulation->read( "dump" + binaryFormat );
```

- Periodically store of a simulation:

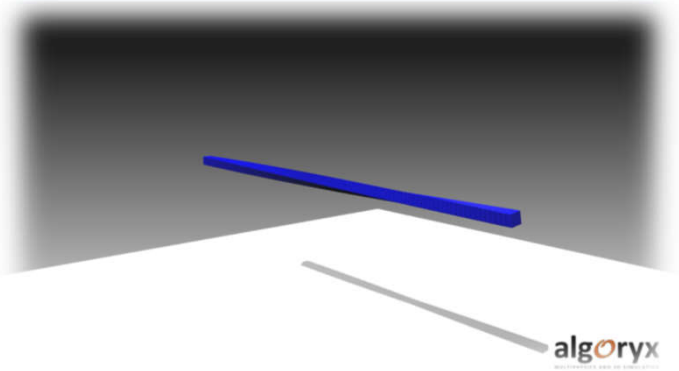
```
// Enable periodical serialization to file.
simulation->getSerializer()->setEnable( true );
// 10 Hz dump interval.
simulation->getSerializer()->setInterval( 0.1 );
// Files will be dump_0001.agx, dump_0002.agx, etc.
simulation->getSerializer()->setFilename( "dump.agx" );
```

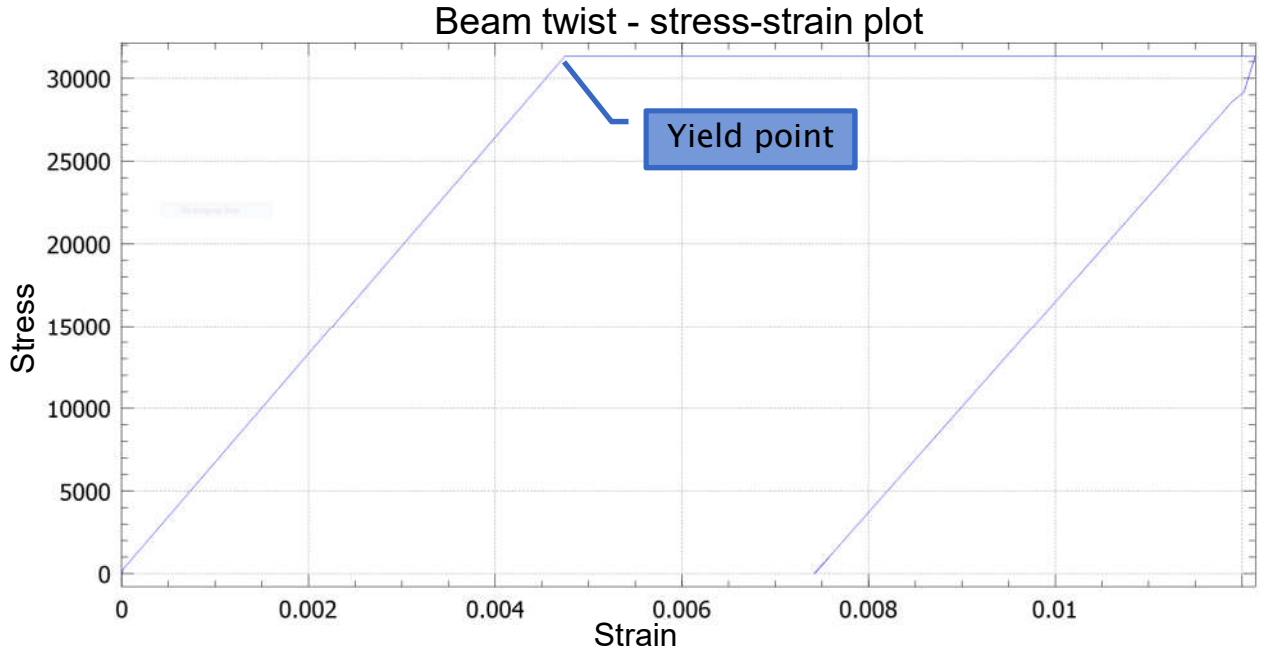
- Disabling serialization for large objects
  - Height fields, large triangle meshes etc.

```
// Read height data from original source instead of
// having it in the agx-file.
heightFieldGeometry->setEnableSerialization( false );
```

agxModel::Deformable1D

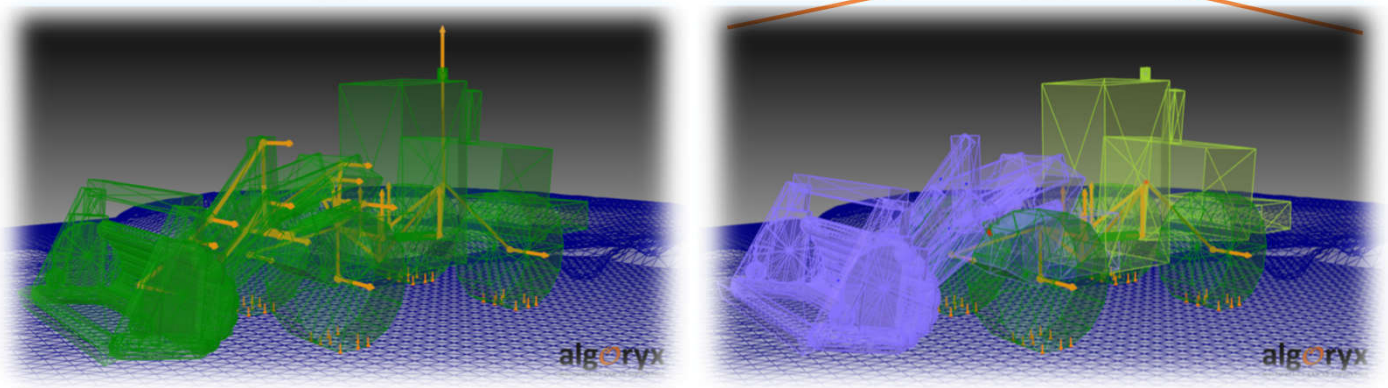
- "One dimensional" six DOF lumped element model.
  - Beam
  - Crane boom
- Routing similar to agxCable::Cable.
- Elasto-plastic deformation.
  - Yield- and break point





agx::MergedBody

- One single rigid body carrying the dynamics properties of several other bodies.
  - Mass
  - Inertia
  - Damping
  - Added mass
  - External forces and torques
  - …



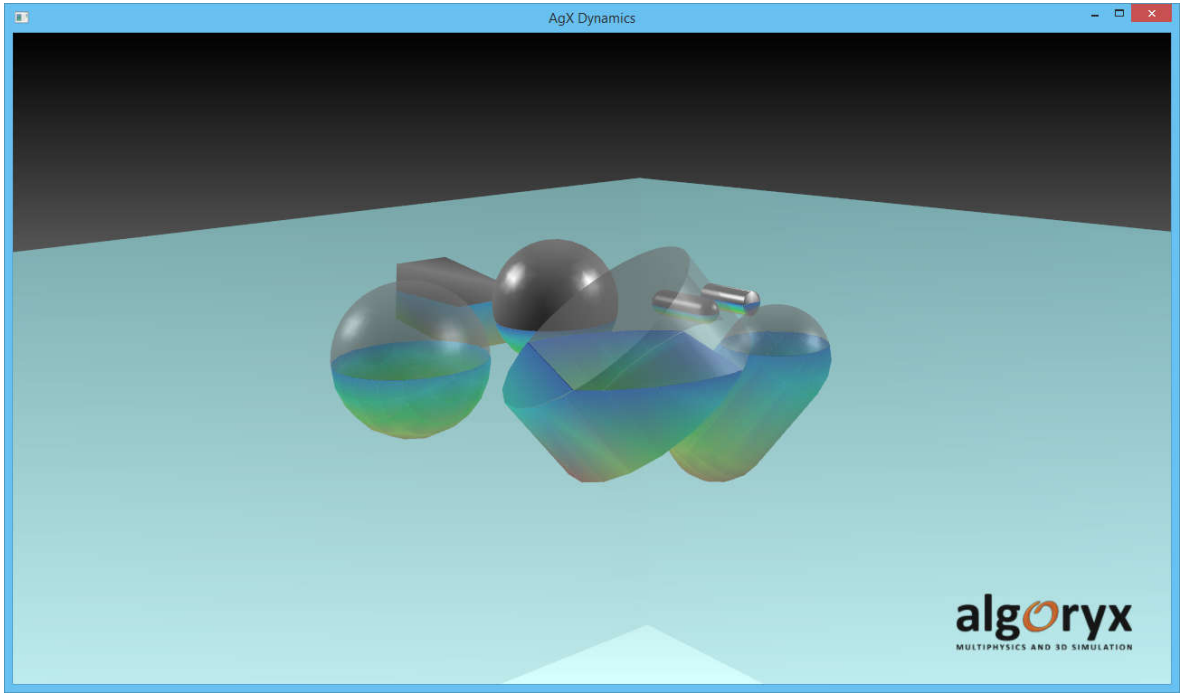
agxSDK::MergeSplitHandler

- Functionality for merging/splitting bodies to reduce the complexity in a simulation.
- Manages agx::MergedBody objects.
- Has access to the complete interaction graph.
- Highly extensible:
  - Split when constraint is under stress
  - Split when tension in a wire exceeds some threshold
  - Split when a motor is activated
  - Merge when a range is activated
  - …

Merge split

Hydro- and aerodynamics

- Surface integral over any shape type.
- Includes:
  - Buoyancy
  - Pressure drag
  - Viscous drag
  - Lift
  - Added mass



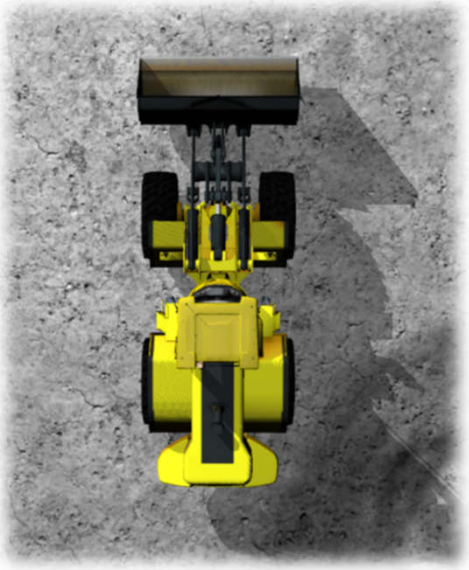
- As an alternative, user methods for hydro- and aerodynamics can be connected via co-simulation (as mentioned earlier).

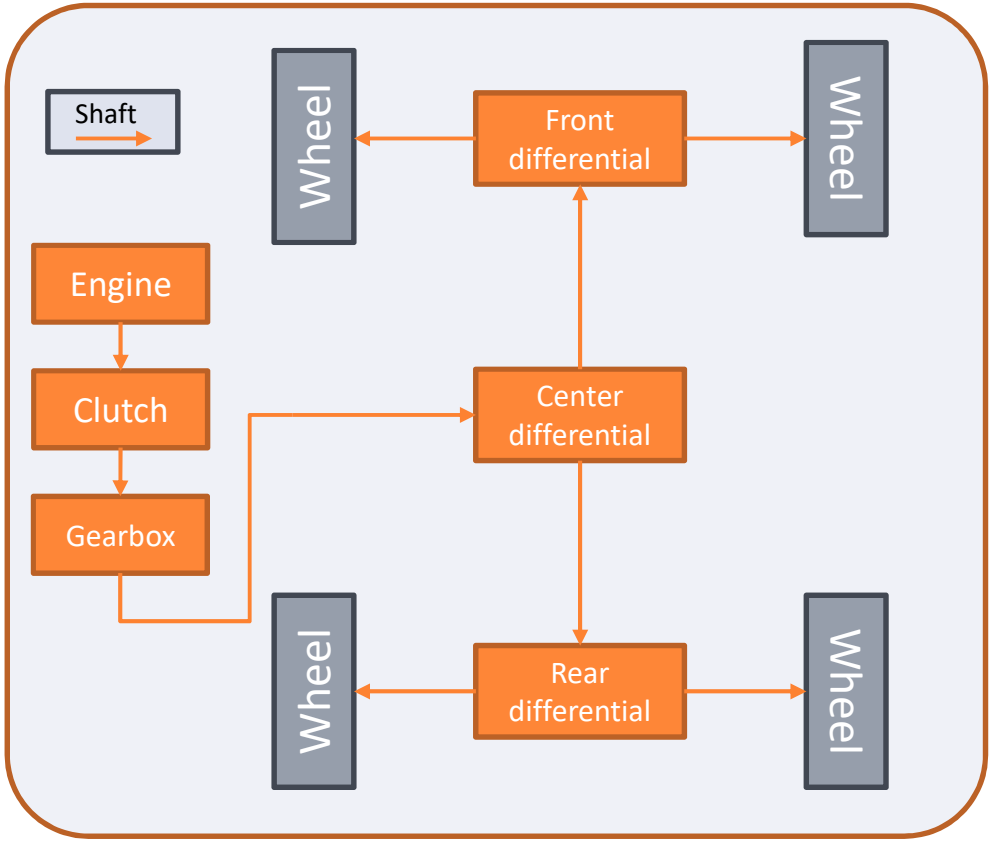
Hydrodynamics

```
// Let the water geometry to be an box of size 140x140x60 meters.
  agxCollide::GeometryRef waterGeometry = new agxCollide::Geometry(new agxCollide::Box(70,
70, 30));

  agxModel::WindAndWaterControllerRef controller = new agxModel::WindAndWaterController();
  controller->addWater(waterGeometry);
  simulation->add(controller);

  simulation->add(waterGeometry);

  // Find various objects in the loaded simulation
  agx::RigidBody *leftWing = simulation->getRigidBody("LeftWing");

  // Set viscous drag for all shapes of the relevant bodies.
  // Pressure drag and lift can be set the same way.
  agxModel::WindAndWaterParameters::setHydrodynamicCoefficient(controller, leftWing,
agxModel::WindAndWaterParameters::VISCOUS_DRAG, 0.01);
```

agxDriveTrain

- Constraint based drive train with generalized components.
  - Electrical and combustion engines
  - Gearboxes
  - Differentials
  - Shafts
  - Torque converters
  - Use cases: vehicles, but also robots





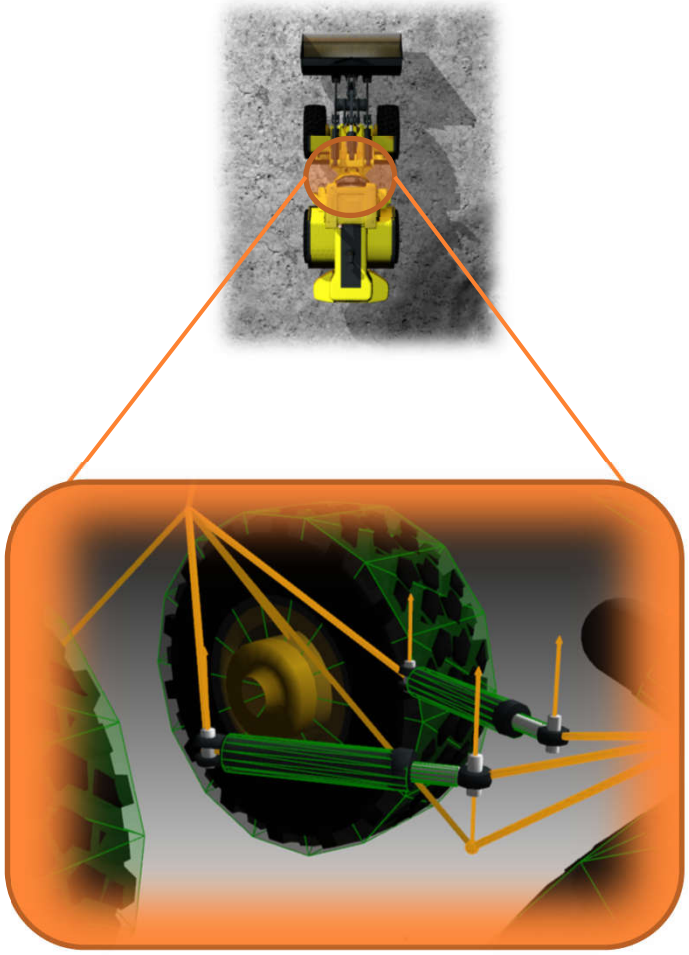
agxHydraulics

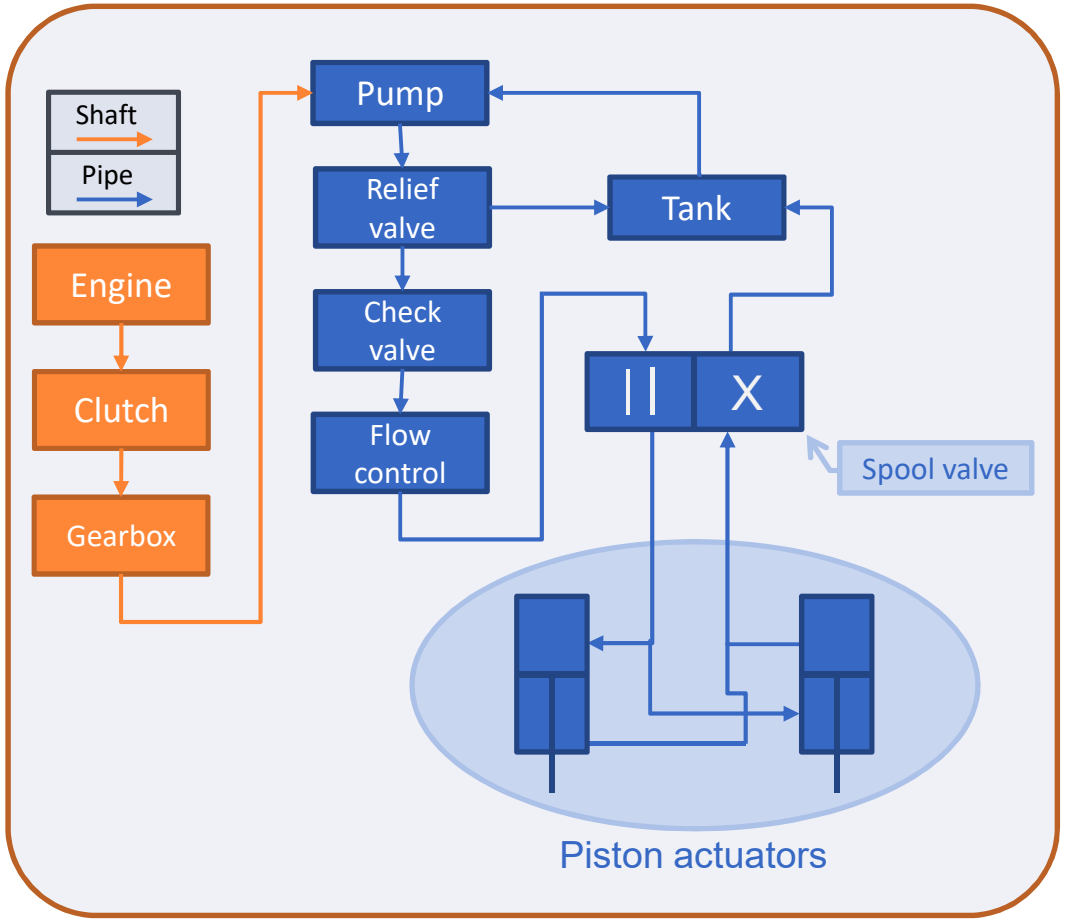
- Constraint based hydraulics components.
  - Advantage compared to using different program: No co-simulation (small time step, oscillations); mechanics and hydraulics solved in one matrix.
- Complex schemes in interactive simulations.
  - Stable at large time steps (e.g., 60 Hz)
- Connect a hydraulic pump/motor to shafts/pipes.
  - Power the hydraulics with the power source from the drive line.
  - Power the drive line with a power source from the hydraulics.
- Instant system response to pressure fluctuations.
  - Transients

HYDRAULIC COMPONENTS

Component	Symbol	Component	Symbol
Accumulator		Fixed displacement pump	
Binary valve		Needle valve	
Check valve		Relief valve	
Constant flow valve		Spool valve	
Cylinder (single rod, double acting)		Variable displacement pump	
Fixed displacement motor		Variable displacement motor	

agxHydraulics – steering a wheel loader





FMI – Functional Mock-up Interface

- Model exchange and co-simulation standard.
- Export FMUs from AGX:
  - Using agxLua (Lua scripting language) and a serialized AGX simulation (scene).
  - Using an XML file format that binds different API calls to FMU model variables.

```
<?xml version="1.0" ?>
<model description="This module simulates a spinning box controlled by a motorized hinge.">
  <variable name="targetMotorSpeed"
            description=""
            targetUuid="c73d6b7b-e0c3-418d-9352-21d6a8b4792f"
            targetMethod="motor.speed"
            causality="input"  variability="continuous" startValue=1.5 />
  <variable name="boxAngularVelocityZ"
            description=""
            targetUuid="9746a517-5591-41e5-8c96-f228cd414200"
            targetMethod="angularVelocity.z"
            causality="output"
            variability="continuous" />
</model>
```

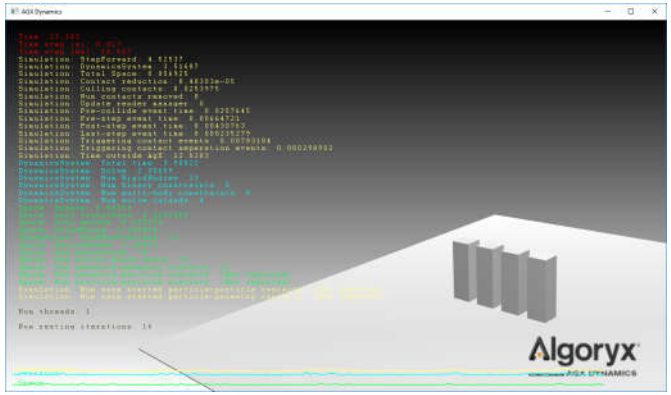
```
for _, constraint in ipairs( robot.constraints ) do
  local motor = constraint:getMotor1D()
  local name = "TargetMotorSpeed_" .. constraint:getName()
  local speedInput = agxFMI1.Export.LuaInputVariable_Real( name )
  speedInput:setStartValue( motor:getSpeed() )

  function speedInput:set( value )
    motor:setSpeed( value )
  end
  module:registerVariable(speedInput)
end
```

agxModel::Terrain

- Simulates a deformable terrain.
  - Tire tracks.
  - Excavator can dig in the soil and fill the bucket with material.
  - Terrain can have varying firmness.

agx::Statistics

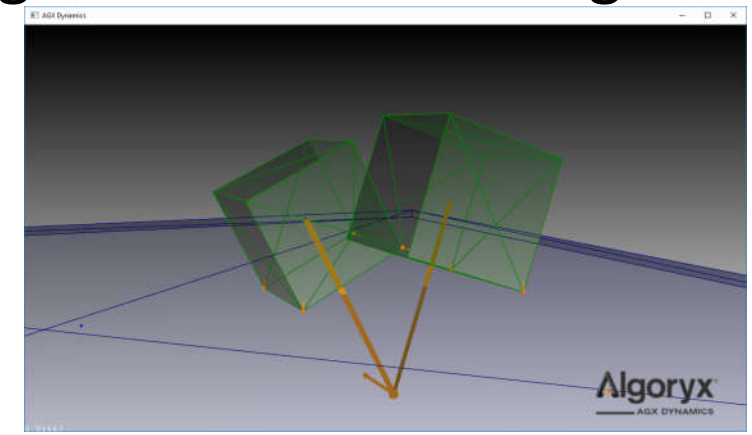
- AGX collects statistics about the simulation
  - Number of geometries, time spent in solver, collision detection, listeners etc.
- agxViewer
  - ’b’ – Toggle the statistics
  - ’n’ – Write statistics to console



- Can be enabled through environment variables or through the API (See relevant chapter in User Manual PDF).

Debug rendering

- An API for overloading customer rendering of the Physics simulation.
  - Geometries/shapes
  - Center of mass
  - Contacts
  - Constraints

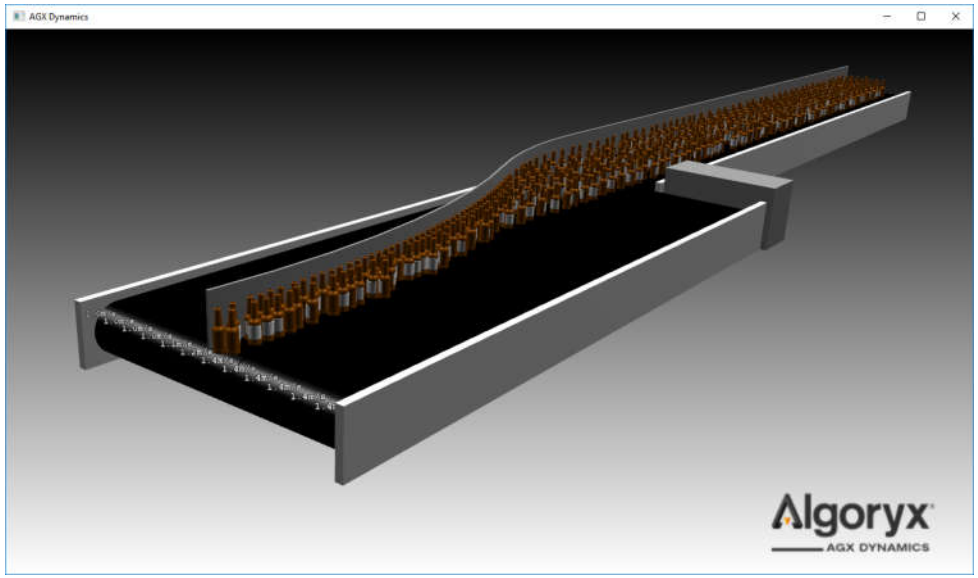


- agxRender:: namespace. See documentation for more information. (Chapter 24 in User Manual)
- An example of this can be seen in the agxOSG library using OpenSceneGraph.

Parallel iterative solver

- Parallel Gauss-Seidel solver
- Faster than single-threaded (if using several threads), but not deterministic
- From agxViewer/ExampleApplication: `agxViewer --useParallelPGS 1 <other arguments>`
- From source code:

```
simulation->getSolver()->setUseParallelPgs(true);
// Also tell agx to use more of the system threads;
in this case, 4:
agx::setNumThreads(4);

// Also note: set up contact system with iterative
solver!
agx::FrictionModelRef frictionModel = new
agx::IterativeProjectedConeFriction();
frictionModel-
>setSolveType(agx::FrictionModel::ITERATIVE);
// For all combinations material1, material2:
cm =
agxSDK::MaterialManager::getOrCreateContactMaterial
(material1, material2);
cm->setFrictionModel(frictionModel);
```



C#

- Part of the API is exported through SWIG (www.swig.org) to C# .NET.
- Can be located (in full code) at <agx>/swig.
- Not a full 1:1 match yet.

Appendix 1

Short overview of Lua programming with AGX

Lua - AGX

- Lua (www.lua.org)
- Compact, fast, yet powerful scripting language.
- Simple to integrate.
- Resembles C/C++ (somewhat).
- For AGX the API is 85% the same.
- Containers/iterators is where it does not bridge well.
- Used internally for testing/prototyping at Algoryx.
- Trend: more test/development done in Python as scripting language, however

C++ vs Lua syntax

- Comments:
  - C++: //, /* */
  - Lua: --, --[[--]]
- Construction:
  - C++: agx::RigidBodyRef body = new agx::RigidBody();
  - Lua: local body = agx.RigidBody()
- Calling a normal method:
  - C++: body->getMassProperties()->setMass(10);
  - Lua: body:getMassProperties():setMass(10)
- Calling a static method:
  - C++: agxOSG::SimulationObject::createBox("box", agx::AffineMatrix4x4(), agx::Vec3(1,1,1), simulation);
  - Lua: agxOSG.SimulationObject.createBox("box", agx.AffineMatrix4x4 (), agx.Vec3(1,1,1), simulation)
- Using an enum:
  - C++: body->setMotionControl(agx::PhysicalBody::STATIC);
  - Lua: body:setMotionControl(agx.PhysicalBody.STATIC)
- Object oriented access of a class pointer to itself:
  - C++: this->setPosition(agx::Vec3());
  - Lua: self:setPosition(agx.Vec3())

Sample script

- From Lua: requestPlugin(”agxOSG”)
  - Extends standard Lua with the AGX API.
- Script can be started using agxViewer (or luaagx).
  - agxViewer myScript.agxLua will search for and call the function: *buildScene(simulation, application)*
  - luaagx myScript.agxLua will just execute the file ”as is”

```
requestPlugin("agxOSG") -- Load the agxOSG script plugin

-- This function will be called when creating the scene
function buildScene(sim, app)
  local root = agxOSG.Group:new()

  -- Create a RigidBody
  local body = agx.RigidBody()

  -- Create a Geometry with a Sphere of radius 0.4
  local geometry = agxCollide.Geometry( agxCollide.Sphere(0.4))
  body:add(geometry) -- Add the geometry to the RigidBody
  sim:add( body)     -- Add the body (and the Geometry to the Simulation)
  body:setPosition( agx.Vec3(1,2,3) ) -- Set the position of the body

  -- Create a visual representation of the geometry
  agxOSG.createVisual(body, root)

  return root // Return this root back to the ExampleApplication
end
```

Creating objects

- We are calling C++ code, objects are allocated on the heap with **new**.
- Question is just, who is responsible for deallocating the memory?
- **RigidBody:new()** – Allocate an object, no reference counting, Lua will NOT try to deallocate memory in any way.
- **RigidBody:new_local()** – Lua will keep an eye on the object and call *collect* when it goes out of scope.
  - This is default for all AGX objects such as RigidBody, Constraint etc. So there, new_local() is not needed; it is enough to write RigidBody().
  - If a class is derived from agx::Referenced
    - new_local() increments ref-count.
  - If a class is NOT derived from agx::Referenced
    - Lua is completely responsible for deallocating it.

Creating objects

- Default allocation (without new/new_local)
  - **local v3 = agx.Vec3(1,2,3)**
  - Same as:
    - **local v3 = agx.Vec3:new_local(1,2,3)**
- Stack allocated objects should ALWAYS be allocated using new_local (just dont use new/new_local):
  - Vec3, Vec4, Quat, AffineMatrix4x4, EulerAngles etc.
- Reference counted objects can be allocated with new/new_local: RigidBody, Geometry, Shape, Simulation, Constraint etc...
- OBS: agxOSG::Node, agxOSG::Group is NOT derived from agx::Referenced, NEVER USE new_local on them.

To summarize

		Example

:new_local	Allocate memory on the heap. If it is a class derived from agx::Referenced, it will increment the reference count. When scope of variable goes out, it will decrement reference count.	`do`
		`  local v1 = agx.Vec3:new_local()`
		`end`
		`-- Scope of v1 goes out, v1 is`
		`-- deallocated`
		`do`
		`  local b = agx.RigidBody:new_local()`
		`end`
		`-- Scope of b goes out, v1 is`
		`-- deallocated`

:new	Allocate memory on the heap, responsibility of deallocating memory is left to C++.	`do`
		`  local b = agx.RigidBody:new()`
		`  -- Simulation is now responsible`
		`  -- for deallocating`
		`  -- (when refcount goes to 0)`
		`  simulation:add( b )`
		`end`

default	Same as **new_local**	`do`
		`  -- Allocate a Vec3, de-allocated`
		`  -- when function call is done`
		`  body:setVelocity(agx.Vec3())`
		`end`

Use it from LUA

```
local sim = agxSDK.Simulation()

local se = agxSDK.LuaStepEventListener()
se:setName("StepListenerExample")
se:setMask(agxSDK.StepEventListener.ALL)
se.sum = 0
sim:add(se)


function se:pre( t )
  print(string.format("PRE> %f: %s\n", t,
self:getName()))
  self.sum = self.sum+1
end
```

How to start/execute Lua code?

- luaagx.exe
  - Extension of code from Lua standard command-line interpreter (lua.exe).
  - Requires you to create an agxOSG::ExampleApplication if graphics are needed.
  - Added functionality for plugins.
  - Many of the scripts in data/luaDemos/tutorials can be started both with agxViewer and luaagx.
- luaagx tests/all_tests.lua
  - Execute the Lua code directly.
  - Create simulation/objects/files.
  - Currently no connection to graphics.
  - Arguments in table: **arg**, (num arguments **#arg**)

Appendix 2

Short overview of Python programming with AGX

Python - AGX

- Python (www.python.org)
- Compact, fast, yet powerful scripting language.
- Simple to integrate
- In general similar to lua, but differences:
  - Very well adapted in scientific community for scientific computing, in some areas replacing Matlab
  - Can be used “stand-alone”, only including agx as a library (lua has to be built as luaagx in order to handle AGX)
  - Large number of external libraries, for 3D visualization, graph-drawing, numerical methods, …

Python – AGX limitations

AGX Python

- AGX Python is only available for the 64-bit version of AGX Dynamics.
- Currently version 3.5 or later of Python is required to use AGX Python under Windows. Under Linux version 3.0 or later is required.

General python:

- Functions and other scopes have to be indented with 4 spaces!

```
# Algoryx Dynamics imports
import agxPython

def buildScene():
    # Get sim, app and root from agxPython context.
    sim = agxPython.getContext().environment.getSimulation()
```

C++ vs Python syntax

- Comments:
  - C++: //, /* */
  - Python: #, no “official” multiline comments
- Construction:
  - C++: agx::RigidBodyRef body = new agx::RigidBody();
  - Python: body = agx.RigidBody()
- Calling a normal method:
  - C++: body->getMassProperties()->setMass(10);
  - Python: body.getMassProperties().setMass(10)
- Calling a static method:
  - C++: agx::Constraint::calculateFramesFromBody(agx::Vec3(1,2,3), agx::Vec3(1,0,0), body1, frame1, body2, frame2);
  - Python: agx.Constraint.calculateFramesFromBody(agx.Vec3(1,2,3), agx.Vec3(1,0,0), body1, frame1, body2, frame2)
- Using an enum:
  - C++: body->setMotionControl(agx::PhysicalBody::STATIC);
  - Python: body.setMotionControl(agx.PhysicalBody.STATIC)
- Object oriented access of a class pointer to itself:
  - C++: this->setPosition(agx::Vec3());
  - Python: self.setPosition(agx.Vec3())

Sample script in agxViewer

- From Python: `import agxPython`
  - Extends standard Python with the AGX API.
  - Has to be done for all used namespaces of AGX as well!
- Script can be started using agxViewer (or python).
- Loading with agxViewer:
  - agxViewer myScript.agxPy will search for and call the function: *buildScene()*

```
# Algoryx Dynamics imports
import agxPython
import agx
import agxCollide
import agxOSG

# Import python modules
import random
import sys

def buildScene():
    # Get sim, app and root from agxPython context.
    sim = agxPython.getContext().environment.getSimulation()
    app = agxPython.getContext().environment.getApplication()
    root = agxPython.getContext().environment.getSceneRoot()
    # Create a Geometry with a Sphere of radius 0.4
    geometry = agxCollide.Geometry( agxCollide.Sphere(0.4))
    body = agx.RigidBody(geometry) # create a rigid body, add the geometry
    sim.add( body)     # Add the body (and the geometry) to the Simulation
    body.setPosition( agx.Vec3(0.1,0.2,0.5) ) # Set the position of the body
    agxOSG.createVisual(body, root) # create graphical representation of the
body
```

Sample script using python-executable

- Loading with python:
  - Same as for agxViewer, but will run file and main-function
  - python myScript.agxPy will execute the file as it is.
- We can find out in script if it is called from agxViewer or python, and make it work for both.

```
# Algoryx Dynamics imports
import agxPython
import agx
import agxCollide
import agxOSG

# Import python modules
import random
import sys

def buildScene():
    # Get
    sim = agxPython.getContext().environment.getSimulation()
    app = agxPython.getContext().environment.getApplication()
    root = agxPython.getContext().environment.getSceneRoot()
    # Create a Geometry with a Sphere of radius 0.4
    geometry = agxCollide.Geometry( agxCollide.Sphere(0.4))
    body = agx.RigidBody(geometry) # create a rigid body, add the geometry
    sim.add( body)     # Add the body (and the geometry) to the Simulation
    body.setPosition( agx.Vec3(0.1,0.2,0.5) ) # Set the position of the body
    agxOSG.createVisual(body, root) # create graphical representation of the
body

def main(args):
    # Create an application with graphics etc.
    app = agxOSG.ExampleApplication()

    # Create a command line parser. sys.executable will point to python
    # executable in this case, because getArgumentName(0) needs to match the
# C argv[0] which  is the name of the program running
    argParser = agxIO.ArgumentParser([sys.executable] + args)

    app.addScene(argParser.getArgumentName(1), "buildScene", ord('1'))

    # Call the init method of ExampleApplication
    # It will setup the viewer, windows etc.
    if app.init(argParser):
        app.run()
    else:
        print("An error occurred while initializing ExampleApplication.")

# Entry point when this script is loaded with python
if agxPython.getContext() is None:
    init = agx.AutoInit()
    main(sys.argv)
```

Event listeners

```
# Now we want to do a task every timestep
# For this we use a agxSDK::StepEventListener. But we want to
implement some virtual methods
class ForceListener(agxSDK.StepEventListener):
    def __init__(self, app, constraints):
        # We give our listener some member attributes
        super().__init__()
        self.app = app
        self.constraints = constraints

    # We implement the post() method
    # which will be triggered after the solver is done.
    # See the documentation of agxSDK::StepEventListener
    # for more virtual methods
    def post(self, t):
        # Do something with constraints...

        # Then, call parent method from c++
        return super().post(t)

def buildScene1(sim, app, root):
    # create constraints ...

    # Now forcelistener:
    forceListener = ForceListener(app, constraints)
```

- In C++, StepEventListeners and ContactEventlisteners overload virtual methods.
- In Python, this is done via virtual methods as well, as seen in the left.
- Note the self (own class) vs. super (parent class) calls