# FLEX

API Reference v0.23

Sat Nov 22 2014

# Contents

# 0.1 File Index

## 0.1.1 File List

Here is a list of all files with brief descriptions:



# 0.2 File Documentation

## 0.2.1 flex.h File Reference

### Data Structures

- struct FlexParams
- struct FlexSDFShape
- struct FlexTimers

### Typedefs

- typedef void(∗ FlexErrorCallback )(const char ∗msg, const char ∗file, int line)

### Enumerations

- enum FlexError { flexSuccess = 0, flexErrorWrongVersion = 1, flexErrorInsufficientGPU = 2 }
- enum FlexMemory { eFlexHost, eFlexDevice, eFlexHostAsync, eFlexDeviceAsync }

### Functions

- FLEX_API FlexError flexInit (int version=FLEX_VERSION, FlexErrorCallback errorFunc=NULL)
- FLEX_API void flexShutdown ()
- FLEX_API int flexGetVersion ()
- FLEX_API FlexSolver ∗ flexCreateSolver (int maxParticles, int maxDiffuseParticles)
- FLEX_API void flexDestroySolver (FlexSolver ∗s)
- FLEX_API void flexUpdateSolver (FlexSolver ∗s, float dt, int substeps, FlexTimers ∗timers)
- FLEX_API void flexSetParams (FlexSolver ∗s, const FlexParams ∗params)
- FLEX_API void flexSetActive (FlexSolver ∗s, const int ∗indices, int n, FlexMemory source)
- FLEX_API void flexGetActive (FlexSolver ∗s, int ∗indices, FlexMemory target)
- FLEX_API int flexGetActiveCount (FlexSolver ∗s)
- FLEX_API void flexSetParticles (FlexSolver ∗s, const float ∗p, int n, FlexMemory source)
- FLEX_API void flexGetParticles (FlexSolver ∗s, float ∗p, int n, FlexMemory target)
- FLEX_API void flexGetSmoothParticles (FlexSolver ∗s, float ∗p, int n, FlexMemory target)
- FLEX_API void flexSetVelocities (FlexSolver ∗s, const float ∗v, int n, FlexMemory source)
- FLEX_API void flexGetVelocities (FlexSolver ∗s, float ∗v, int n, FlexMemory target)
- FLEX_API void flexSetPhases (FlexSolver ∗s, const int ∗phases, int n, FlexMemory source)
- FLEX_API void flexGetPhases (FlexSolver ∗s, int ∗phases, int n, FlexMemory target)
- FLEX_API void flexSetSprings (FlexSolver ∗s, const int ∗indices, const float ∗restLengths, const float ∗stiffness, int numSprings, FlexMemory source)
- FLEX_API void flexGetSprings (FlexSolver ∗s, int ∗indices, float ∗restLengths, float ∗stiffness, int numSprings, FlexMemory target)
- FLEX_API void flexSetRigids (FlexSolver ∗s, const int ∗offsets, const int ∗indices, const float ∗restPositions, const float ∗restNormals, const float ∗stiffness, const float ∗rotations, int numRigids, FlexMemory source)
- FLEX_API void flexSetNormals (FlexSolver ∗s, const float ∗normals, int n, FlexMemory source)

- FLEX_API void flexGetNormals (FlexSolver *s, float *normals, int n, FlexMemory target)
- FLEX_API void flexGetRigidTransforms (FlexSolver *s, float *rotations, float *translations, FlexMemory target)
- FLEX_API void flexSetConvexes (FlexSolver *s, const float *aabbMin, const float *aabbMax, const int *planeOffsets, const int *planeCounts, const float *planes, const float *positions, const float *rotations, const int *flags, int numConvexes, int numPlanes, FlexMemory source)
- FLEX_API void flexSetTriangles (FlexSolver *s, const int *indices, const float *vertices, int numTris, int numVertices, float cellSize, FlexMemory source)
- FLEX_API void flexSetShapes (FlexSolver *s, const FlexSDFShape *shapes, int numShapes)
- FLEX_API void flexSetDynamicTriangles (FlexSolver *s, const int *indices, const float *normals, int numTris, FlexMemory source)
- FLEX_API void flexGetDynamicTriangles (FlexSolver *s, int *indices, float *normals, int numTris, FlexMemory target)
- FLEX_API void flexSetInflatables (FlexSolver *s, const int *startTris, const int *numTris, float *restVolumes, float *overPressures, float *constraintScales, int numInflatables, FlexMemory source)
- FLEX_API void flexGetDensities (FlexSolver *s, float *densities, FlexMemory target)
- FLEX_API void flexGetAnisotropy (FlexSolver *s, float *q1, float *q2, float *q3, FlexMemory target)
- FLEX_API int flexGetDiffuseParticles (FlexSolver *s, float *p, float *v, int *indices, FlexMemory target)
- FLEX_API void flexSetDiffuseParticles (FlexSolver *s, const float *p, const float *v, int n, FlexMemory source)
- FLEX_API void flexGetBounds (FlexSolver *s, float *lower, float *upper)
- FLEX_API void * flexAlloc (int size)
- FLEX_API void flexFree (void *ptr)
- FLEX_API void flexSetFence ()
- FLEX_API void flexWaitFence ()

#### 0.2.1.1 Data Structure Documentation

##### 0.2.1.1.1 struct FlexParams

Simulation parameters for a solver

**Data Fields**

| | | |
|---|---|---|
| | FlexRelaxation-Mode | |

**Data Fields**

| | | |
|---|---|---|
| int | mNumIterations | Number of solver iterations to perform per-substep. |
| float | mGravity[3] | Constant acceleration applied to all particles. |
| float | mRadius | The maximum interaction radius for particles. |
| float | mSolidRest-Distance | The distance non-fluid particles attempt to maintain from each other, must be in the range (0, radius]. |
| float | mFluidRest-Distance | The distance fluid particles are spaced at the rest density, must be in the range (0, radius], for fluids this should generally be 50-70% of mRadius, for rigids this can simply be the same as the particle radius. |
| float | mDynamic-Friction | Coefficient of friction used when colliding against shapes. |
| float | mStaticFriction | Coefficient of static friction used when colliding against shapes. |
| float | mParticleFriction | Coefficient of friction used when colliding particles. |
| float | mRestitution | Coefficient of restitution used when colliding against shapes, particle collisions are always inelastic. |

| | | |
|---|---|---|
| float | mAdhesion | Controls how strongly particles stick to surfaces they hit, default 0.0, range [0.0, +inf]. |
| float | mSleep-Threshold | Particles with a velocity magnitude $<$ this threshold will be considered fixed. |
| float | mMaxVelocity | Particle velocity will be clamped to this value at the end of each step. |
| float | mShock-Propagation | Artificially decrease the mass of particles based on height from a fixed reference point, this makes stacks and piles converge faster. |
| float | mDissipation | Damp particle velocity based on how many particle contacts it has. |
| float | mEnableCCD | If true then a second collision detection pass will be executed against triangle meshes to prevent tunneling, usually not necessary, only enable if having tunnelling problems. |
| float | mWind[3] | Constant acceleration applied to particles that belong to dynamic triangles, drag needs to be $>$ 0 for wind to affect triangles. |
| float | mDrag | Drag force applied to particles belonging to dynamic triangles, proportional to velocity$^\wedge$2$*$area in the negative velocity direction. |
| float | mLift | Lift force applied to particles belonging to dynamic triangles, proportional to velocity$^\wedge$2$*$area in the direction perpendicular to velocity and (if possible), parallel to the plane normal. |
| bool | mFluid | If true then particles with phase 0 are considered fluid particles and interact using the position based fluids method. |
| float | mCohesion | Control how strongly particles hold each other together, default: 0.025, range [0.0, +inf]. |
| float | mSurface-Tension | Controls how strongly particles attempt to minimize surface area, default: 0.0, range: [0.0, +inf]. |
| float | mViscosity | Smoothes particle velocities using XSPH viscosity. |
| float | mVorticity-Confinement | Increases vorticity by applying rotational forces to particles. |
| float | mAnisotropy-Scale | Control how much anisotropy is present in resulting ellipsoids for rendering. |
| float | mSmoothing | Control the strength of Laplacian smoothing in particles for rendering, see flexGetSmoothParticles() |
| float | mSolidPressure | Add pressure from solid surfaces to particles. |
| float | mFreeSurface-Drag | Drag force applied to boundary fluid particles. |
| float | mBuoyancy | Gravity is scaled by this value for fluid particles. |
| float | mDiffuse-Threshold | Particles with kinetic energy + divergence above this threshold will spawn new diffuse particles. |
| float | mDiffuse-Buoyancy | Scales force opposing gravity that diffuse particles receive. |
| float | mDiffuseDrag | Scales force diffuse particles receive in direction of neighbor fluid particles. |
| int | mDiffuseBallistic | The number of neighbors below which a diffuse particle is considered ballistic. |
| float | mDiffuseSort-Axis[3] | Diffuse particles will be sorted by depth along this axis if non-zero. |
| float | mPlastic-Threshold | Particles belonging to rigid shapes that move with a position delta magnitude $>$ threshold will be permanently deformed in the rest pose. |
| float | mPlasticCreep | Controls the rate at which particles in the rest pose are deformed for particles passing the deformation threshold. |
| float | mParticle-CollisionMargin | Increases the radius used during neighbor finding, this is useful if particles are expected to move significantly during a single step to ensure contacts aren't missed on subsequent iterations. |

| | | |
|---|---|---|
| float | mShape-CollisionMargin | Increases the radius used during contact finding against kinematic shapes. |
| float | mCollision-Distance | Distance particles maintain against shapes. |
| float | mPlanes[8][4] | Collision planes in the form ax + by + cz + d = 0. |
| int | mNumPlanes | Num collision planes. |
| FlexRelaxation-Mode | mRelaxation-Mode | How the relaxation is applied inside the solver. |
| float | mRelaxation-Factor | Control the convergence rate of the parallel solver, default: 1, values greater than 1 may lead to instability. |

#### 0.2.1.1.2 struct FlexSDFShape

Signed distance field collision shape, note that only cubic fields are currently supported (mWidth=mHeight=mDepth)

**Data Fields**

| | | |
|---|---|---|
| float | mLower[3] | Shape AABB lower bounds in world space. |
| float | mUpper[3] | Shape AABB upper bounds in world space. |
| float | mInvEdge-Length[3] | 1/(mUpper-mLower) |
| unsigned int | mWidth | |
| unsigned int | mHeight | Field x dimension in voxels. |
| unsigned int | mDepth | Field y dimension in voxels. |
| const float * | mField | Field z dimension in voxels. |

#### 0.2.1.1.3 struct FlexTimers

Time spent in each section of the solver update, times in seconds, see flexUpdateSolver()

**Data Fields**

| | | |
|---|---|---|
| float | mPredict | Time spent in prediction. |
| float | mCreateCell-Indices | Time spent creating grid indices. |
| float | mSortCell-Indices | Time spent sorting grid indices. |
| float | mCreateGrid | Time spent creating grid. |
| float | mReorder | Time spent reordering particles. |
| float | mCollide-Particles | Time spent finding particle neighbors. |
| float | mCollide-Convexes | Time spent colliding convex shapes. |
| float | mCollide-Triangles | Time spent colliding triangle shapes. |
| float | mCollideFields | Time spent colliding signed distance field shapes. |
| float | mCalculate-Density | Time spent calculating fluid density. |
| float | mSolveDensities | Time spent solving density constraints. |
| float | mSolveVelocities | Time spent solving velocity constraints. |
| float | mSolveShapes | Time spent solving rigid body constraints. |
| float | mSolveSprings | Time spent solving distance constraints. |

| | | |
|---|---|---|
| float | mSolveContacts | Time spent solving contact constraints. |
| float | mSolve-Inflatables | Time spent solving pressure constraints. |
| float | mCalculate-Anisotropy | Time spent calculating particle anisotropy for fluid. |
| float | mUpdateDiffuse | Time spent updating diffuse particles. |
| float | mUpdate-Triangles | Time spent updating dynamic triangles. |
| float | mUpdate-Normals | Time spent updating vertex normals. |
| float | mFinalize | Time spent finalizing state. |
| float | mUpdateBounds | Time spent updating particle bounds. |
| float | mTotal | Sum of all timers above. |

#### 0.2.1.2 Typedef Documentation

##### 0.2.1.2.1 typedef void(* FlexErrorCallback)(const char *msg, const char *file, int line)

Error reporting callback.

#### 0.2.1.3 Enumeration Type Documentation

##### 0.2.1.3.1 enum FlexError

Flex error types

**Enumerator**

***flexSuccess*** The API call returned with no errors.

***flexErrorWrongVersion*** The header version does not match the library binary.

***flexErrorInsufficientGPU*** The GPU associated with the calling thread does not meet requirements. An SM3.0 GPU or above is required.

##### 0.2.1.3.2 enum FlexMemory

Designates a memory space for getting/settings data to/from

**Enumerator**

***eFlexHost*** Host (CPU) memory.

***eFlexDevice*** Device (GPU) memory.

***eFlexHostAsync*** Host (CPU) memory asynchronous, when used with a flexGet/flexSet method the memory transfer will be asynchronous and should be synchronized with flexWaitFence()

***eFlexDeviceAsync*** Device (GPU) memory asynchronous, when used with a flexGet/flexSet method the memory transfer will be asynchronous and should be synchronized with flexWaitFence()

#### 0.2.1.4 Function Documentation

##### 0.2.1.4.1 FLEX_API FlexError flexInit ( int *version* = FLEX_VERSION, FlexErrorCallback *errorFunc* = NULL )

Initialize library, should be called before any other API function.

**Note**

Flex uses the calling thread's CUDA context for all operations so users should make sure the same context is used for all API calls (this should be the case if just using the default runtime context).

#### 0.2.1.4.2 FLEX_API void flexShutdown ( )

Shutdown library, users should manually destroy any previously created solvers to ensure memory is freed before calling this method.

#### 0.2.1.4.3 FLEX_API int flexGetVersion ( )

Get library version number

#### 0.2.1.4.4 FLEX_API FlexSolver* flexCreateSolver ( int *maxParticles,* int *maxDiffuseParticles* )

Create a new particle solver

**Parameters**

| | | |
|---|---|---|
| in | *maxParticles* | Maximum number of simulation particles possible for this solver |
| in | *maxDiffuse-Particles* | Maximum number of diffuse (non-simulation) particles possible for this solver |

#### 0.2.1.4.5 FLEX_API void flexDestroySolver ( FlexSolver * *s* )

Delete a particle solver

#### 0.2.1.4.6 FLEX_API void flexUpdateSolver ( FlexSolver * *s,* float *dt,* int *substeps,* **FlexTimers** * *timers* )

Move particle solver forward in time

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *dt* | Time to integrate the solver forward in time by |
| in | *substeps* | The time dt will be divided into the number of sub-steps given by this parameter |
| out | *timers* | If non-NULL this struct will be filled out with profiling information for the step, note that profiling can substantially slow down overal performance so this param should only be non-NULL in non-release builds |

#### 0.2.1.4.7 FLEX_API void flexSetParams ( FlexSolver * *s,* const **FlexParams** * *params* )

Update solver paramters

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *params* | Parameters structure in host memory, see FlexParams |

#### 0.2.1.4.8 FLEX_API void flexSetActive ( FlexSolver * *s,* const int * *indices,* int *n,* **FlexMemory** *source* )

Set the active particles indices in the solver

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *indices* | Holds the indices of particles that have been made active |
| in | *n* | Number of particles to allocate |
| in | *source* | The memory space of the indices |

#### 0.2.1.4.9 FLEX_API void flexGetActive ( FlexSolver * *s,* int * *indices,* **FlexMemory** *target* )

Return the active particle indices

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *indices* | An array of indices at least activeCount in length |
| in | *target* | The memory space of the destination buffer |

#### 0.2.1.4.10 FLEX_API int flexGetActiveCount ( FlexSolver * *s* )

Return the number of active particles in the solver

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |

**Returns**

The number of active particles in the solver

#### 0.2.1.4.11 FLEX_API void flexSetParticles ( FlexSolver * *s*, const float * *p*, int *n*, FlexMemory *source* )

Set the particles state of the solver, a particle consists of 4 floating point numbers, it's x,y,z position followed by it's inverse mass (1/m)

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *p* | Pointer to an array of particle data, should be 4*n in length |
| in | *n* | The number of particles to set |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.12 FLEX_API void flexGetParticles ( FlexSolver * *s*, float * *p*, int *n*, FlexMemory *target* )

Get the particles state of the solver, a particle consists of 4 floating point numbers, it's x,y,z position followed by it's inverse mass (1/m)

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *p* | Pointer to an array of 4*n floats that will be filled out with the particle data, can be either a host or device pointer |
| in | *n* | The number of particles to get, must be less than max particles passed to flexCreateSolver |
| in | *target* | The memory space of the destination buffer |

#### 0.2.1.4.13 FLEX_API void flexGetSmoothParticles ( FlexSolver * *s*, float * *p*, int *n*, FlexMemory *target* )

Get the Laplacian smoothed particle positions for rendering, see FlexParams::mSmoothing

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *p* | Pointer to an array of 4*n floats that will be filled out with the data, can be either a host or device pointer |
| in | *n* | The number of smooth particles to return |
| in | *target* | The memory space of the destination buffer |

#### 0.2.1.4.14 FLEX_API void flexSetVelocities ( FlexSolver * *s*, const float * *v*, int *n*, FlexMemory *source* )

Set the particle velocities, each velocity is a 3-tuple of x,y,z floating point values

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *v* | Pointer to an array of 3∗n floats |
| in | *n* | The number of velocities to set |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.15 FLEX_API void flexGetVelocities ( FlexSolver ∗ *s*, float ∗ *v*, int *n*, FlexMemory *target* )

Get the particle velocities, each velocity is a 3-tuple of x,y,z floating point values

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *v* | Pointer to an array of 3∗n floats that will be filled out with the data, can be either a host or device pointer |
| in | *n* | The number of velocities to get |
| in | *target* | The memory space of the destination buffer |

#### 0.2.1.4.16 FLEX_API void flexSetPhases ( FlexSolver ∗ *s*, const int ∗ *phases*, int *n*, FlexMemory *source* )

Set the particles phase id array, each particle has an associated phase id which controls how it interacts with other particles. Particles with phase 0 interact with all other phase types.

Particles with a non-zero phase id only interact with particles whose phase differs from theirs. This is useful, for example, to stop particles belonging to a single rigid shape from interacting with each other.

Phase 0 is used to indicate fluid particles when FlexParams::mFluid is set.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *phases* | Pointer to an array of n integers containing the phases |
| in | *n* | The number of phases to set |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.17 FLEX_API void flexGetPhases ( FlexSolver ∗ *s*, int ∗ *phases*, int *n*, FlexMemory *target* )

Get the particle phase ids

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *phases* | Pointer to an array of n integers that will be filled with the phase data, can be either a host or device pointer |
| in | *n* | The number of phases to get |
| in | *target* | The memory space of the destination buffer |

#### 0.2.1.4.18 FLEX_API void flexSetSprings ( FlexSolver ∗ *s*, const int ∗ *indices*, const float ∗ *restLengths*, const float ∗ *stiffness*, int *numSprings*, FlexMemory *source* )

Set spring constraints for the solver. Each spring consists of two particle indices stored consecutively, a rest-length, and a stiffness value.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |

| | | |
|---|---|---|
| in | *indices* | Pointer to the spring indices array, should be 2∗numSprings length, 2 indices per-spring |
| in | *restLengths* | Pointer to an array of rest lengths, should be numSprings length |
| in | *stiffness* | Pointer to the spring stiffness coefficents, should be numSprings in length, a negative stiffness value represents a tether constraint |
| in | *numSprings* | The number of springs to set |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.19 FLEX_API void flexGetSprings ( FlexSolver ∗ *s,* int ∗ *indices,* float ∗ *restLengths,* float ∗ *stiffness,* int *numSprings,* FlexMemory *target* )

Get the spring constraints for the solver

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *indices* | Pointer to the spring indices array, should be 2∗numSprings length, 2 indices per-spring |
| out | *restLengths* | Pointer to an array of rest lengths, should be numSprings length |
| out | *stiffness* | Pointer to the spring stiffness coefficents, should be numSprings in length, a negative stiffness value represents a unilateral tether constraint (only resists stretching, not compression), valid range [-1, 1] |
| in | *numSprings* | The number of springs to get |
| in | *target* | The memory space of the destination buffers |

#### 0.2.1.4.20 FLEX_API void flexSetRigids ( FlexSolver ∗ *s,* const int ∗ *offsets,* const int ∗ *indices,* const float ∗ *restPositions,* const float ∗ *restNormals,* const float ∗ *stiffness,* const float ∗ *rotations,* int *numRigids,* FlexMemory *source* )

Set rigid body constraints for the solver.

**Note**

A particle should not belong to more than one rigid body at a time.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *offsets* | Pointer to an array of start offsets for a rigid in the indices array, should be numRigids+1 in length, the first entry must be 0 |
| in | *indices* | Pointer to an array of indices for the rigid bodies, the indices for the jth rigid body start at indices[offsets[j]] and run to indices[offsets[j+1]] exclusive |
| in | *restPositions* | Pointer to an array of local space positions relative to the rigid's center of mass (average position), this should be at least 4∗numIndices in length in the format x,y,z,w |
| in | *restNormals* | Pointer to an array of local space normals, this should be at least 4∗numIndices in length in the format x,y,z,w where w is the (negative) signed distance of the particle inside it's shape |
| in | *stiffness* | Pointer to an array of rigid stiffness coefficents, should be numRigids in length, valid values in range [0, 1] |
| in | *rotations* | Pointer to an array of 3x3 rotation matrices (9∗numRigids in length) |
| in | *numRigids* | The number of rigid bodies to set |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.21 FLEX_API void flexSetNormals ( FlexSolver ∗ *s,* const float ∗ *normals,* int *n,* FlexMemory *source* )

Set per-particle normals to the solver, these will be overwritten after each simulation step

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *normals* | Pointer to an array of normals, should be 4∗n in length |
| in | *n* | The number of normals to set |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.22 FLEX_API void flexGetNormals ( FlexSolver ∗ *s,* float ∗ *normals,* int *n,* FlexMemory *target* )

Get per-particle normals from the solver, these are the world-space normals computed during surface tension and rigid body calculations

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *normals* | Pointer to an array of normals, should be 4∗n in length |
| in | *n* | The number of normals to get |
| in | *target* | The memory space of the destination buffer |

#### 0.2.1.4.23 FLEX_API void flexGetRigidTransforms ( FlexSolver ∗ *s,* float ∗ *rotations,* float ∗ *translations,* FlexMemory *target* )

Get the rotation matrices for the rigid bodies in the solver

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *rotations* | Pointer to an array of 3x3 rotation matrices to hold the rigid rotations, should be 9∗numRigids floats in length |
| out | *translations* | Pointer to an array of vectors to hold the rigid translations, should be 3∗numRigids floats in length |
| in | *target* | The memory space of the destination buffer |

#### 0.2.1.4.24 FLEX_API void flexSetConvexes ( FlexSolver ∗ *s,* const float ∗ *aabbMin,* const float ∗ *aabbMax,* const int ∗ *planeOffsets,* const int ∗ *planeCounts,* const float ∗ *planes,* const float ∗ *positions,* const float ∗ *rotations,* const int ∗ *flags,* int *numConvexes,* int *numPlanes,* FlexMemory *source* )

Set the convex collision shapes for the solver, the convex data is specified in structure of array (SOA) format.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *aabbMin* | Point to an array of lower AABB coordinates for each convex in world space, should be 4∗numConvexes in length in x,y,z,∗ format |
| in | *aabbMax* | Pointer to an array of upper AABB coordinates for each convex in world space, should be 4∗numConvexes in length in x,y,z,∗ format |
| in | *planeOffsets* | Pointer to an array of start offsets into the planes array for each convex, should be numConvexes in length |
| in | *planeCounts* | Pointer to an array of counts representing the number of planes belonging to each convex, should be numConvexes in length |
| in | *planes* | Pointer to an array of planes defining the convex shapes in ax + by + cz + d = 0 form, planes are specified in a local coordinate solver, should be 4∗numPlanes in length |

| | | |
|---|---|---|
| in | *positions* | Pointer to an array of translations for each convex in world space, should be 4∗numConvexes in length |
| in | *rotations* | Pointer to an an array of rotations for each convex stored as quaternion, should be 4∗numConvexes in length |
| in | *flags* | Whether the convex is considered static (0), or dynamic (1) |
| in | *numConvexes* | The number of convexes |
| in | *numPlanes* | The total number of planes for all convexes |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.25 FLEX_API void flexSetTriangles ( FlexSolver ∗ *s,* const int ∗ *indices,* const float ∗ *vertices,* int *numTris,* int *numVertices,* float *cellSize,* FlexMemory *source* )

Set the triangle mesh collision data for the solver, triangles are treated as two-sided and collided using a continuous collison detection method to prevent tunnelling

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *indices* | Pointer to an array of triangle vertex indices, should be 3∗numTris in length |
| in | *vertices* | Pointer to an array of vertex positions in world space, should be 3∗numPositions in length |
| in | *numTris* | The number of triangles |
| in | *numVertices* | The number of mesh vertices |
| in | *cellSize* | The size of grid cell used for broad phase collision culling, should be set relative to particle radius, e.g. 2∗radius |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.26 FLEX_API void flexSetShapes ( FlexSolver ∗ *s,* const FlexSDFShape ∗ *shapes,* int *numShapes* )

Set the signed distance field collision shapes, see FlexSDFShape.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *shapes* | Pointer to an array of signed distance field shapes |
| in | *numShapes* | The number of shapes |

#### 0.2.1.4.27 FLEX_API void flexSetDynamicTriangles ( FlexSolver ∗ *s,* const int ∗ *indices,* const float ∗ *normals,* int *numTris,* FlexMemory *source* )

Set dynamic triangles mesh indices, typically used for cloth. Flex will calculate their normals and apply wind and drag effects to connected particles. See FlexParams::mDrag, FlexParams::mWind.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *indices* | Pointer to an array of triangle indices into the particles array, should be 3∗numTris in length |
| in | *normals* | Pointer to an array of triangle normals, should be 3∗numTris in length, can be NULL |
| in | *numTris* | The number of dynamic triangles |
| in | *source* | The memory space of the source buffers |

#### 0.2.1.4.28 FLEX_API void flexGetDynamicTriangles ( FlexSolver ∗ *s,* int ∗ *indices,* float ∗ *normals,* int *numTris,* FlexMemory *target* )

Get the dynamic triangle indices and normals.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *indices* | Pointer to an array of triangle indices into the particles array, should be 3∗numTris in length, if NULL indices will not be returned |
| out | *normals* | Pointer to an array of triangle normals, should be 3∗numTris in length, if NULL normals will be not be returned |
| in | *numTris* | The number of dynamic triangles |
| in | *target* | The memory space of the destination arrays |

#### 0.2.1.4.29 FLEX_API void flexSetInflatables ( FlexSolver ∗ *s,* const int ∗ *startTris,* const int ∗ *numTris,* float ∗ *restVolumes,* float ∗ *overPressures,* float ∗ *constraintScales,* int *numInflatables,* FlexMemory *source* )

Set inflatable shapes, an inflatable is a range of dynamic triangles that represent a closed mesh. Each inflatable has a given rest volume, constraint scale (roughly equivalent to stiffness), and "over pressure" that controls how much the shape is inflated.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| in | *startTris* | Pointer to an array of offsets into the solver's dynamic triangles for each inflatable, should be numInflatables in length |
| in | *numTris* | Pointer to an array of triangle counts for each inflatable, should be numInflatablesin length |
| in | *restVolumes* | Pointer to an array of rest volumes for the inflatables, should be numInflatables in length |
| in | *overPressures* | Pointer to an array of floats specifying the pressures for each inflatable, a value of 1.0 means the rest volume, $>$ 1.0 means over-inflated, and $<$ 1.0 means under-inflated, should be numInflatables in length |
| in | *constraintScales* | Pointer to an array of scaling factors for the constraint, this is roughly equivalent to stiffness, see helper code for details, should be numInflatables in length |
| in | *numInflatables* | Number of inflatables to set |
| in | *source* | The memory space of the source buffers |

#### 0.2.1.4.30 FLEX_API void flexGetDensities ( FlexSolver ∗ *s,* float ∗ *densities,* FlexMemory *target* )

Get the density values for fluid particles

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *densities* | Pointer to an array of floats, should be maxParticles in length, density values are normalized between [0, 1] where 1 represents the rest density |
| in | *target* | The memory space of the destination arrays |

#### 0.2.1.4.31 FLEX_API void flexGetAnisotropy ( FlexSolver ∗ *s,* float ∗ *q1,* float ∗ *q2,* float ∗ *q3,* FlexMemory *target* )

Get the anisotropy of fluid particles, the particle distribution for a particle is represented by 3 orthogonal vectors. Each 3-vector has unit length with the variance along that axis packed into the w component, i.e.: x,y,z,lambda.

The anisotropy defines an oriented ellipsoid in worldspace that can be used for rendering or surface extraction.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *q1* | Pointer to an array of floats that receive the first basis vector and scale, should be 4∗maxParticles in length |

| | | |
|---|---|---|
| out | *q2* | Pointer to an array of floats that receive the second basis vector and scale, should be 4∗maxParticles in length |
| out | *q3* | Pointer to an array of floats that receive the third basis vector and scale, should be 4∗maxParticles in length |
| in | *target* | The memory space of the destination arrays |

#### 0.2.1.4.32 FLEX_API int flexGetDiffuseParticles ( FlexSolver ∗ *s,* float ∗ *p,* float ∗ *v,* int ∗ *indices,* FlexMemory *target* )

Get the state of the diffuse particles. Diffuse particles are passively advected by the fluid velocity field.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *p* | Pointer to an array of floats, should be 4∗maxParticles in length, the w component represents the particles lifetime with 1 representing a new particle, and 0 representing an inactive particle |
| out | *v* | Pointer to an array of floats, should be 4∗maxParticles in length, the w component is not used |
| out | *indices* | Pointer to an array of ints that specify particle indices in depth sorted order, should be maxParticles in length, see FlexParams::mDiffuseSortDir |
| in | *target* | The memory space of the destination arrays |

#### 0.2.1.4.33 FLEX_API void flexSetDiffuseParticles ( FlexSolver ∗ *s,* const float ∗ *p,* const float ∗ *v,* int *n,* FlexMemory *source* )

Set the state of the diffuse particles. Diffuse particles are passively advected by the fluid velocity field.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *p* | Pointer to an array of floats, should be 4∗n in length, the w component represents the particles lifetime with 1 representing a new particle, and 0 representing an inactive particle |
| out | *v* | Pointer to an array of floats, should be 4∗n in length, the w component is not used |
| in | *n* | Number of diffuse particles to set |
| in | *source* | The memory space of the source buffer |

#### 0.2.1.4.34 FLEX_API void flexGetBounds ( FlexSolver ∗ *s,* float ∗ *lower,* float ∗ *upper* )

Get the world space AABB of all particles in the solver.

**Parameters**

| | | |
|---|---|---|
| in | *s* | A valid solver |
| out | *lower* | Pointer to an array of 3 floats to receive the lower bounds |
| out | *upper* | Pointer to an array of 3 floats to receive the upper bounds |

#### 0.2.1.4.35 FLEX_API void∗ flexAlloc ( int *size* )

Allocates size bytes of memory from the optimal memory pool. Using this function is optional, but when passed to flexGet/flexSet methods it may result in significantly faster transfers, memory used with async transfers should be allocated by this method to ensure optimal performance. For CUDA implementations this method will return pinned host memory from cudaMallocHost().

**Parameters**

| | | |
|---|---|---|
| in | *size* | The number of bytes to alloc |

**Returns**

pointer to the allocated memory

#### 0.2.1.4.36 FLEX_API void flexFree ( void * *ptr* )

Free memory allocated through flexAlloc

**Parameters**

| | | |
|---|---|---|
| in | *ptr* | Pointer returned from flexAlloc |

#### 0.2.1.4.37 FLEX_API void flexSetFence ( )

Sets a fence that can be used to synchronize the calling thread with any outstanding GPU work, typically used with eFlexHostAsync to ensure any flexGet/flexSet calls have completed.

```
// update solver
flexUpdateSolver(solver, dt, iterations, NULL);

// read back state
flexGetParticles(solver, &particleBuffer, n, flexHostAsync);
flexGetVelocities(solver, &velocityBuffer, n, flexHostAsync);
flexGetDensities(solver, &densityBuffer, n, flexHostAsync);

// insert fence
flexSetFence();

// perform asynchronous CPU work

// wait for queued work to finish
flexWaitFence();
```

#### 0.2.1.4.38 FLEX_API void flexWaitFence ( )

Wait's for the work scheduled before the last call to flexSetFence() to complete If flexSetFence() has not yet been called then this is function returns immediately

# Index