Dynamic Mesh Compression Toolkit

Libor Váša, Oldřich Petřík, Jan Rus

16.3.2010

1 Introduction

This is basic documentation for the Dynamic Mesh Compression Toolkit (DMCT). The aim of the toolkit is to make it easier to implement and test new algorithms for compressing or comparing animations of triangular meshes of shared connectivity (dynamic meshes).

The toolkit works with the MVE-2[FVS06] environment and consists of implementations of several basic and advanced algorithms needed for construction of complex compression algorithms. These algorithms are provided as separate classes, which are used in example MVE-2 modules. The toolkit also contains modules for usual tasks related to dynamic mesh compression, such as loading dynamic meshes from files, saving dynamic meshes into files and comparing dynamic meshes. Finally the toolkit provides a set of MVE-2 maps which demonstrate how to use the modules.

The toolkit links to a GPL linear algebra library Math.NET Iridium[iri].

2 Installation

The installation of the toolkit is done by copying the contents of the distribution (i.e. the `Mve2-bin` and `Mve2-src` folders) over a decompressed version of MVE-2, which can be downloaded from `http://herakles.zcu.cz/research/mve2`. The compression toolkit can be used either by using the modules described in the following sections, or by linking the `Compression.dll` class library to a new project. The source code of the whole toolkit is provided in the `Mve2-src` folder.

3 Data structures

3.1 Dynamic mesh

Dynamic meshes are represented by standard MVE-2 structures *UniformDataArray* and *TriangleMesh*. An array of triangle meshes is interpreted as a sequence of frames with constant temporal distance of 1/25s. Many modules expect frames to share connectivity, and these modules use the connectivity of the first

frame of the sequence. Note that, in order to achieve better performance, modules which expect shared connectivity usually do not perform a check whether the connectivity is indeed equal for all the frames.

4 Provided modules

4.1 AnimationBinaryReader

The module restores a dynamic mesh from a binary file previously created by the `AnimationBinarySaver` module.

4.2 AnimationBinarySaver

The module saves an animation into a binary file. The module does not compress the data, and the resulting file may be used to either quickly restore an animation or to compare a compression algorithm against binary storage. The module saves a single connectivity, i.e. it requires input in the format described in section 3.1.

4.3 AnimationColorMapping

Maps values of a number point attribute present in each frame of the input animation to colors. Outputs the input animation with the colors added to the frames as a point attribute. The resulting mapping is relative to the global range of the number values over the whole animation. A palette module has to be connected to the *OutValue* and *InColor* ports for this module to work. Palette modules can be found in *Zcu.Mve.Visualization* package.

4.4 AnimationVRMLLoader

Loads an animation from a VRML 2.0 file. Performs parsing of the input and provides the result in the form of *UniformDataArray*, which is an array of frames of the animation. Each frame is represented as a *TriangleMesh* structure, which contains points and triangles. The connectivity is shared by all the frames, and it can be accessed from any of the frames.

4.5 AnimationVRMLSaver

Saves a dynamic mesh into a VRML 2.0 file. The animation should be provided in the format described in previous section. The connectivity should be equal for all the frames, however the module does NOT check whether that requirement is met, it directly uses the connectivity of the first frame of the animation.

4.6 AnimationComparerKG

Compares two animations represented by UniformDataArrays (see section 3.1). The comparison is done according to the Karni and Gotsman paper [KG04]. Since the paper is unclear about several issues, the module uses methods used by most later papers in the field. The module computes per-frame averages of values and Frobenius norms of matrices.

4.7 AnimationComparerSTED

Compares two animations represented by UniformDataArrays (see section 3.1). The comparison is done according to the Váša and Skala paper[VS10]. The module requires the distance of two most distant vertices in the first frame, which can be computed by the *Distance* module provided in the standard MVE-2 library *Visualization*.

4.8 AnimationDistancePainter

Measures the distance between corresponding points of each frame of an original version and a distorted version of an animation represented by UniformDataArray (see section 3.1). Outputs the distorted animation with the distances added to each frame as a point attribute of the mesh.

4.9 AnimationDistorter

Distorts the input animation by adding random values to vertex coordinates. The module supports following types of distortion:

1. `Gauss` generates and adds a random Gaussian distributed value for each coordinate of each vertex in each frame.
2. `GaussTemp` generates and adds a random Gaussian distributed value for each coordinate in each frame (vertices in each frame are shifted by the same amount).
3. `GaussConst` generates and adds a random Gaussian distributed value for each coordinate and each vertex (each vertex is shifted by the same amount in all the frames of the animation).
4. `GaussConstLength` works in a way similar to `GaussConst`, but the deviation of the Gaussian values is inversely proportional to the length of edges incident with each vertex.
5. `SinSpat` Adds a sine of each coordinate to each coordinate of each vertex in each frame.
6. `SinTemp` Adds a sine of frame index to each coordinate of each vertex in each frame.

The respective distortion types can be further controlled by module property `Amount` which controls the amplitude of sines or the standard deviation of Gaussian random values, and by module property `Freq` which controls the frequency of sines.

4.10 AnimationSqrDistPainter

Measures the square distance between corresponding points of each frame of an original version and a distorted version of an animation represented by UniformDataArray (see section 3.1). Outputs the distorted animation with the square distances added to each frame as a point attribute of the mesh.

4.11 ArrayLength

Outputs the length of an input UniformDataArray as an Integer number.

4.12 CompressorCoddyac

Compresses a dynamic mesh using PCA in the space of trajectories as suggested by Váša and Skala[VS07]. The PCA basis is encoded by the Cobra[VS09] algorithm. The algorithm takes three inputs:

- the quantization constant for the feature vectors (usually set to 0-4),
- the quantization constant for the PCA basis (usually set to 17-21),
- number of used basis vectors (usually set to 30-150).

The algorithm uses the EdgeBreaker (see sec. 5.4) implementation to traverse the mesh. All the data is saved into a single file, which can be decompressed by the *DecompressorCoddyac* (sec. 4.13) module.

The module provides the resulting data rate (in bits per frame and vertex) as output, along with "simulated" decompression output. The simulated output has geometry equivalent to a real decompression result, however in contrast to decompression result it has the same vertex indices as the input, and thus it can be directly compared against the original input dynamic mesh.

4.13 DecompressorCoddyac

Decompressed a dynamic mesh from a file created by the CompressorCoddyac module (see sec. 4.12). The decompression reconstructs a PCA basis and vertices, using the EdgeBreaker (see sec. 5.4) algorithm to traverse the mesh. Note that due to the nature of the Edgebreaker connectivity compression, the decompression result will have different vertex indices, and as such it **cannot** be directly compared with the original input of the compression. The decompression result however may be compared to the output of the result of the *DummyCompressor* module (see sec. 4.14), which performs reindexing while it preserves the original vertex coordinates.

4.14 DummyCompressor

Performs a simulated compression and decompression using Edgebreaker, so that the resulting mesh has the original geometry, but the indices are changed so that the result can be directly compared to a result of a real decompression.

4.15 MeshComparerMSE

Compares an original version and a distorted version of a static mesh (UnstrGrid or TriangleMesh). Uses a mean square error measure, which calculates the average value of squared distances between corresponding points of the original and distorted mesh.

4.16 MeshComparerVisualKG

Compares an original version and a distorted version of a static mesh (UnstrGrid or TriangleMesh). Uses the *Visual Error* measure described by Karni and Gotsman in [KG00]. Calculates the average of the norm of the geometric distance between the meshes and the norm of the Laplacian difference.

4.17 MeshDistorter

Distorts a static mesh by adding noise to vertices. Supports following types of distortion:

- `Gauss` adds a Gaussian distributed random value to each coordinate of each vertex.
- `Sine` adds sine of each coordinate to each coordinate of each vertex.
- `Uniform` adds a uniformly distributed random value to each coordinate of each vertex.

The standard deviation (res. amplitude) of the distortions can be controlled by the `Amount` property of the module.

5 Provided classes

5.1 ArithCoder

The `ArithCoder` class provides an implementation of context adaptive binary arithmetic coding similar to CABAC[MWS03]. The class provides two important static methods:

- `void encode(int[] data, Stream target)` encodes the array of integers into the provided stream,

- `int[] decode(Stream source)` decodes array of integers from a stream into which the data has been previously written using the `encode` method.

The algorithm internally uses exp-Golomb binarisation scheme, and as such is especially efficient for input data exponentially distributed around zero. The algorithm is able to process negative values.

5.2 UniformQuantizer

The class provides an easy to use interface for uniform quantization and dequantisation of values. An instance of the class is created by calling a constructor which accepts one parameter - the quantization constant, which determines the size of quantization bins. The instance provides two important methods:

- `int Encode(double v)` quantizes the input value using the given quantization constant,
- `double Restore(int v)` restores the original value represented by encoded value v.

5.3 PCA

The PCA class encapsulates an instance of Principal Component Analysis. The class provides methods for finding an uncorrelated basis for a set of d-dimensional samples, finding a representation of a vector in the new (potentially reduced) basis, restoring a sample value in the original space and encoding the basis using the Cobra algorithm[VS09]. The class provides following important methods and fields:

- `double[][] Data` is the input set of samples, each sample expressed as an array of double values.
- `ComputeBasis()` performs the analysis of the data, and initializes the internal data structures computing the new uncorrelated basis and the means vector.
- `double[] GetCoefs(double[] vector, int count)` computes the representation of the provided vector in the new uncorrelated basis. Specified number of coefficients is computed and returned.
- `double[] Restore(double[] vector)` reconstructs the original coordinates of a sample.
- `double EncodeBasisCobra(Stream target, int count, double delta)` encodes the given number of basis vectors into the provided data stream using the provided quatization constant. The method uses the Cobra algorithm to encode the basis.

5.4 EdgebreakerCompressor

Traverses topology of the mesh using the Edgebreaker algorithm[Ros99] and provides compressed information about topology. It also raises events `FirstTriangle` and `NewVertex` associated with geometry compression. The class provides following methods and structures:

- `Stream OutputStream` stream for stored topology description
- `int getNumOfComponents()` returns number of components of the compressed mesh
- `string getCLERS()` returns CLERS topology description of the compressed mesh
- `UniformDataArray getHandles()` returns list of handles of the compressed mesh
- `UniformDataArray getHoles()` returns list of holes of the compressed mesh
- `UniformDataArray getHoleStarts()` returns list of components of the mesh starting by holes
- `int[] getVisitedTriangles()` returns list of triangles of the compressed mesh in order they were visited
- `int[] getO()` returns Edgebreaker O-table
- `int[] getV()` returns Edgebreaker V-table
- `Run(UniformDataArray inputTris, int points)` starts topology compression and raises geometry compression events
- `EventHandler firstTriangleEventHandler` event handler for the first triangle of each component
- `EventHandler newVectorEventHandler` event handler for the new visited vertex

For details on usage of the class see the source code of the `CompressorCoddyac` class.

5.5 EdgebreakerDecompressor

Algorithm reconstructs topology of a mesh from data provided by Edgebreaker[Ros99] compressor. It also raises events `FirstTrianlge` and `NewVertex` associated with geometry decompression. The class provides following methods and structures:

- `Stream InputStream` stream with stored topology description

- `UniformDataArray Run()` starts topology decompression and raises geometry compression events, needs InputStream to be set first
- `UniformDataArray Run(char[] clers, int numOfComponents, int[] holes, int[] holeStarts, int[] handles)` starts topology decompression and raises geometry compression events
- `int[] getInvisibleTriangles()` returns list of invisible triangles of the mesh (eg. trinagles filling holes)
- `int[] getV()` returns Edgebreaker V-table

For details on usage of the class see the source code of the `DecompressorCoddyac` class.

6 Provided example maps

6.1 AnimationPlayer

The map loads a VRML file containing an animation. This animation is iterated in a loop and for each frame vertex normals are computed. Finally, the animation is played back using the SimpleXNARenderer module.

6.2 CompressCoddyac

Compresses a loaded animation using the Coddyac[VS07] algorithm. The data is written into the `data.dat` file.

6.3 CompressCoddyacSTEDCompare

Compresses a loaded animation using the Coddyac[VS07] algorithm. The data is written into the `data.dat` file. The compression result is compared with the original using the STED algorithm. The map demonstrates how to set the inputs for the `AnimationComparerSTED` module.

6.4 DecompressKGCompare

Decompresses the file `data.dat` file and compares the result with the original using the KG error metric. The map demonstrated how to use the DummyCompressor module to reindex vertices of the animation.

6.5 DistortCompareKG

Demonstrates possibilities of artificial distortion module. The distortion result is compared with the original using the KG error metric.

6.6 ErrorPainter

Shows the distribution of introduced error using the `AnimatonDistancePainter` module. The loaded animation is compressed using the Coddyac algorithm and per-vertex compared against the original. Finally, vertices with large distortion are painted red, and vertices with small distortion are painted green.

References

[FVS06] Milan Frank, Libor Váša, and Václav Skala. Mve-2 applied in education process. In *.NET Technologies 2006*, pages 39–45, Plzen, 2006. University of West Bohemia.

[iri] http://www.mathdotnet.com/iridium.aspx.

[KG00] Zachi Karni and Craig Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 279–286, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[KG04] Zachi Karni and Craig Gotsman. Compression of soft-body animation sequences. In *"Computers & Graphics 28, 1"*, pages 25–34, 2004.

[MWS03] Detlev Marpe, Thomas Wiegand, and Heiko Schwarz. Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Trans. Circuits Syst. Video Techn.*, 13(7):620–636, 2003.

[Ros99] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[VS07] Libor Váša and Václav Skala. Coddyac: Connectivity driven dynamic mesh compression. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video*, Kos, Greece, May 2007. IEEE Computer Society.

[VS09] Libor Váša and Václav Skala. Cobra: Compression of the basis for the pca represented animations. *Computer Graphics Forum*, 28(6):1529–1540, 2009.

[VS10] Libor Váša and Václav Skala. A perception correlated comparison method for dynamic meshes. *To appear in IEEE Transactions on Visualization and Computer Graphics*, 2010.