A Scripting language for Digital Content Creation Applications

Mohammed Yousef, Ahmed Hashem, Hassan Saad, Amr Gamal, Osama Galal[*] and Khaled F. Hussain[†]
Faculty of Computers and Information, Assiut University, Assiut, Egypt

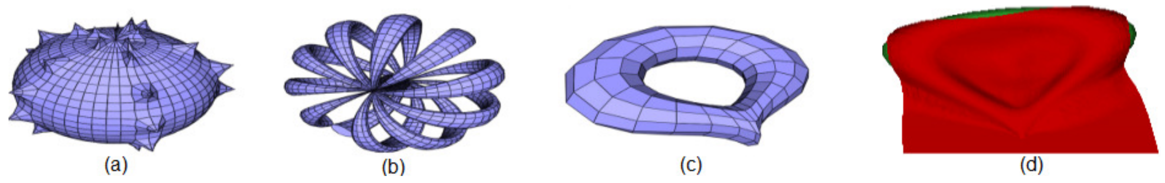

Figure 1: *from left to right, (a) a sphere with bevel on faces of 5 sides then Catmull-Clark subdivision (b) a spring with bend on Y axis (c) a torus with taper on Y axis (d) Cloth(red) covering a torus(green) after tapering it during a physics simulation.*

1 Introduction

Digital Content Creation (DCC) Applications (e.g. Blender, Autodesk 3ds Max) have long been used for the creation and editing of digital content. Due to current advancement in the field, the need for controlled automated work forced these applications to add support for small programming languages that gave power to artists without diving into many details. With time these languages developed into more mature languages and were used for more complex tasks (driving physics simulations, controlling particle systems, or even game engines). For long, these languages have been interpreted, embedded within the applications, lagging the UIs or incomparable with real programming languages (regarding Completeness, Expressiveness, Extensibility and Abstractions). Two approaches were used to implement those languages. Either build them from scratch (like MaxScript), or use an existing popular language and write a set of extensions to it and embed it (like Blender and Python). In practice, both those solutions suffer, the first method produces languages lacking being real, competitive languages and generally very inefficient, the second method has problems arising from not being dedicated in first place for that kind of applications so, they lack expressiveness facilities (like dedicated constructs) that support that particular domain, also it's very hard to optimize these languages for specific DCC situations.

In this paper, we present a system that addresses those problems. A high level scripting language (Zlang) and a DCC Engine, the language can be interpreted, compiled, extended in C/C++ and has a number of constructs and optimizations dedicated to DCC domain. The engine provides geometric primitives, mesh modifiers, key-framed animation and Physics Simulations (Rigid/Soft Body, Cloth and Fluid Simulations), the engine also is designed and implemented as a library so it can be used alone and embedded.

2 TECHNICAL APPROACH

In the DCC engine, we store meshes in a half-edge data structure. Two classes of algorithms for constructing primitives are used, either sculpturing (using Euler operations) or connecting generated point sets into faces. Mesh modifiers operate on half-edges to achieve required effect; they either operate per face (like Extrude and Outline) or on the whole mesh (like Twist and Bend). Every object has a stack of modifiers that are applied to it in order. To produce animation, key-frame values are given to modifier properties, these values are then interpolated so that each property of each modifier has a value at a given frame, applying modifiers on copy of meshes at each frame produces the animation. We use a scene graph data structure for managing the scene.

We provide algorithms for representing our objects accurately as convex shapes in physics simulations and use ACD [ming Lien and Amato 2007] for approximating general shapes to convexes. A unique feature we provide is deducing convex decomposition of our concave primitives depending on their half-edge structure; this allows efficient and accurate representation of complexly modified primitives in physics simulations (e.g. a tube that is bended and twisted to form a pipe, or as used in Figure 1(d)).

Our scripting language is dynamic, memory managed and object oriented (hybrid paradigm). An interpreter is provided for the language with an interest that Zlang scripts be as standalone as possible and cross platform. The interpreter focuses also in optimizing DCC objects creation and array creation and manipulaion.

3 IMPLEMENTATION AND FUTURE WORK

In choosing tools for the system we focused on high performance open source or free, cross platform tools with large communities. We used CGAL Polyhedron [Kettner 2010] for storing meshes. GSL is used for interpolating values for key-framed animation using cubic splines. Eigen libray is used for efficient matrix manipulation. OpenSceneGraph is used for managing the scene graph and other effects (e.g. lights). We used PhysX as a physics engine (we preferred it over Bullet due to its current GPU acceleration). ANTLR [Parr and Quong 1994] is used as a parser and lexer generator, in interpretation; the syntax tree is optimized, traversed and executed. For the future we are now working on two sides. First we are implementing parts of our system in OpenCL (especially global modifiers). Second we are developing a Compiler for Zlang using LLVM as a backend as we are aiming at standalone interactive simulations.

References

KETTNER, L., 2010. 3d polyhedral surfaces. CGAL Editorial Board - 3.6 editions.

MING LIEN, J., AND AMATO, N. M. 2007. Approximate convex decomposition of polyhedra. In *Proceedings of the ACM Symposium on Solid and Physical Modeling*, ACM, 121–131.

PARR, T. J., AND QUONG, R. W. 1994. Antlr: A predicated-ll(k) parser generator. *Software Practice and Experience 25*, 789–810.

[*]e-mail: {mohamed.mahdi, ahmed.ali, hassan.rezg, amr.abdelhafiz, usama.mohamed}@compit.au.edu.eg
[†]e-mail: khaled.hussain2000@gmail.com

Copyright is held by the author / owner(s).
SIGGRAPH 2010, Los Angeles, California, July 25 – 29, 2010.
ISBN 978-1-4503-0210-4/10/0007