Computer Graphics: A Bouncing Jelly Ball

https://huit.re/mpri-jelly

Théophile Bastian and Rémi Oudin

MPRI 2017 – 2018
These slides: https://huit.re/mpri-jelly-slides

2018-14-02

Table of Contents

1 Story Board
2 Algorithms used
3 Implementation
4 Demo Time!

Story Board

A Bouncing Ball

- Oh, noes! Someone dropped their jelly!
- The jelly is a red-ish ball.
- It bounces on the ground.
- A ground can be flat or any smooth curve.
- The ball has an initial speed on x and z axis.

Table of Contents

1 Story Board

2 Algorithms used
- Marching Cubes
- Jelly time!
- Improved Perlin Noise

3 Implementation

4 Demo Time!

Algorithms used

- Focus on low-level implementation
- Jelly described as an implicit surface
  $\rightsquigarrow$ finding a good jelly equation
- Implement Marching Cubes to render it
- Tentative implementation of a non-flat, Perlin-based ground
  $\rightsquigarrow$ broken physics and rendering

Marching Cubes: brief reminder

- Implicit surface $\rightarrow$ triangulated mesh

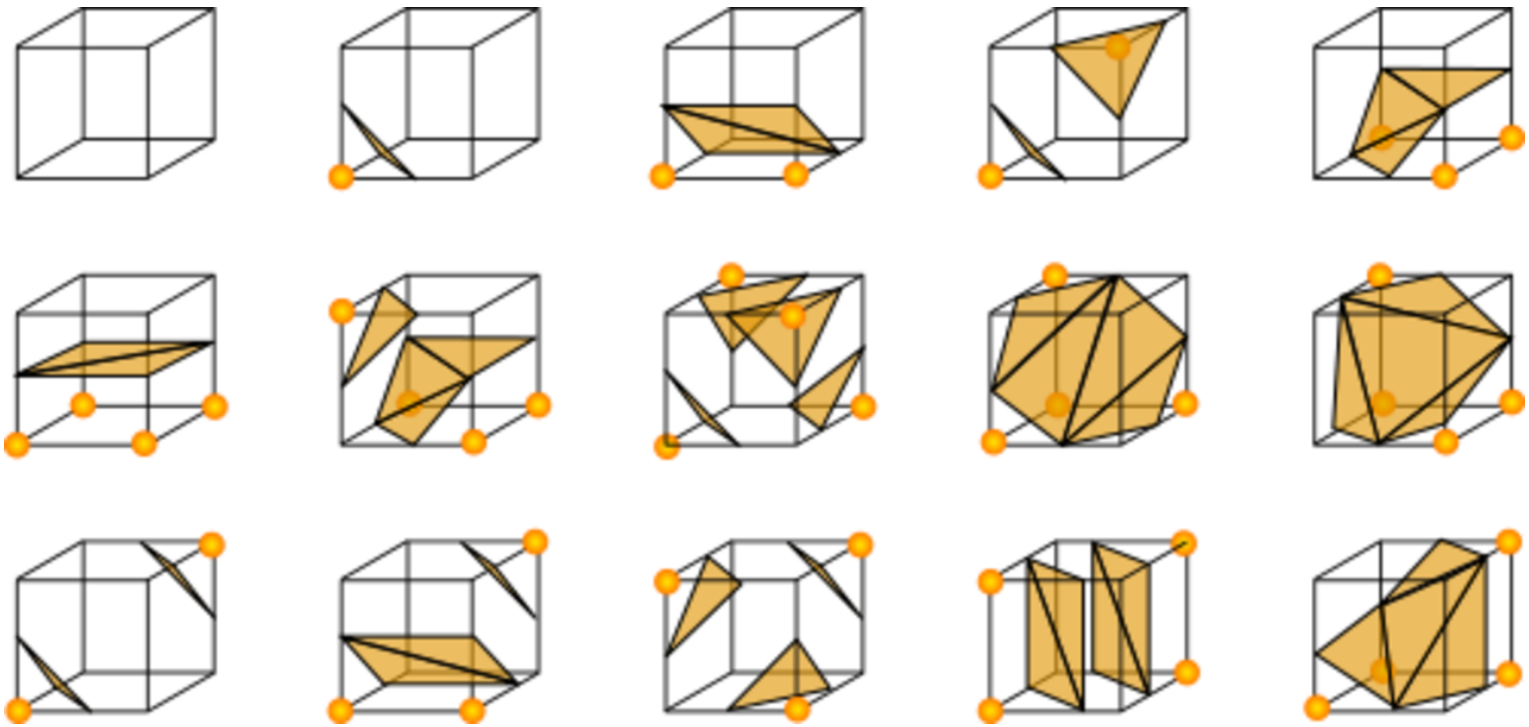
Algorithm 1: Marching cubes (naive)

---

for *each elementary cube c in space* **do**
  Compute whether cube vertices are inside the volume;
  $\rightsquigarrow$ 256 possible configurations;
  Case analysis: add corresponding faces and vertices;

- Actually, 15 configurations up to rotations, symmetries
- Yield edges on which vertices go
- Binary search: actual vertice position along the edge

Well-known configurations



Source: wikipedia, under GNU GPLv2, by Jean-Marie Favreau

Marching Cubes: details

- Configurations generation: with python, generating a C++ file
- Explored space: bounded with a user-provided bounding box
- User-provided hint: BFS from this point to find an intersection, then BFS from this intersection to follow the surface
- Could probably be avoided by binary-searching-ish method to find first intersection


I'M MAD JELLY

Modelling jelly

- Represented as a spheroid of radius q and height p.
- By default it is a sphere
- Volume is constant inside the box.
- When bouncing on the ground, continuously deforms by updating $p = d(ground, center)$ and $q = \sqrt{3/4 \cdot V/(\pi * p)}$
- The maximal deformation of the ball is bounded by a given parameter.
- Implementation: The ball has a physical model, and embeds the implicit surface as an attribute.

Improved Perlin Noise

Brief reminder

- Algorithm to generate pseudo random texture.
- Here used for ground generation.
- Used with a Fractional Brownian motion sum for better results.
- Implemented as an Implicit Surface: perlin(x, y, z) returns 0 if (x, y, z) is in the surface.
- In essence $perlin(x, y, z) = y - fBm(x, 0, z)$, thus y is the value of the noise at ordinate 0.

Improved Perlin Noise

Improved you said?

- Original algorithm had 2 deficiencies : Discontinuity in second order interpolation and non-optimal gradient computation.
- The interpolation function changes from $3t^2 - 2t^3$ to $6t^5 - 15t^4 + 10t^{31}$:
- Gradient function isn't random anymore, the vectors are the one from the directions of the center of a cube to its edges. The direction is the result of P the permutation vector, modulo 12.
- The permutation vector can be randomly generated from a seed.
- A lot of work in order to find «good» parameters.

[1] `http://mrl.nyu.edu/~perlin/paper445.pdf`

Table of Contents

1 Story Board

2 Algorithms used

3 Implementation

4 Demo Time!

Implementation overview

- Guidelines remainder: “use whatever open-source lib you want, but the more it becomes overkill, the more your demo must be awesome”

Implementation overview

- Guidelines remainder: “use whatever open-source lib you want, but the more it becomes overkill, the more your demo must be awesome”
- . . . or, when rephrased: “use whatever open-source lib you want, but the duller it is, the more your demo is allowed to look incredibly bad”

Implementation overview

- Guidelines remainder: “use whatever open-source lib you want, but the more it becomes overkill, the more your demo must be awesome”
- . . . or, when rephrased: “use whatever open-source lib you want, but the duller it is, the more your demo is allowed to look incredibly bad”
- Thus, we used raw OpenGL with GLU and GLUT
- Nothing more, pure C++

Implementation overview

- Guidelines remainder: “use whatever open-source lib you want, but the more it becomes overkill, the more your demo must be awesome”
- . . . or, when rephrased: “use whatever open-source lib you want, but the duller it is, the more your demo is allowed to look incredibly bad”
- Thus, we used raw OpenGL with GLU and GLUT
- Nothing more, pure C++
- You’ve been warned.

Implementation: details

- 2000 SLOC
- Loads of code to make OpenGL work
- Lighting is still broken (how does that even work?!)

Table of Contents

1 Story Board

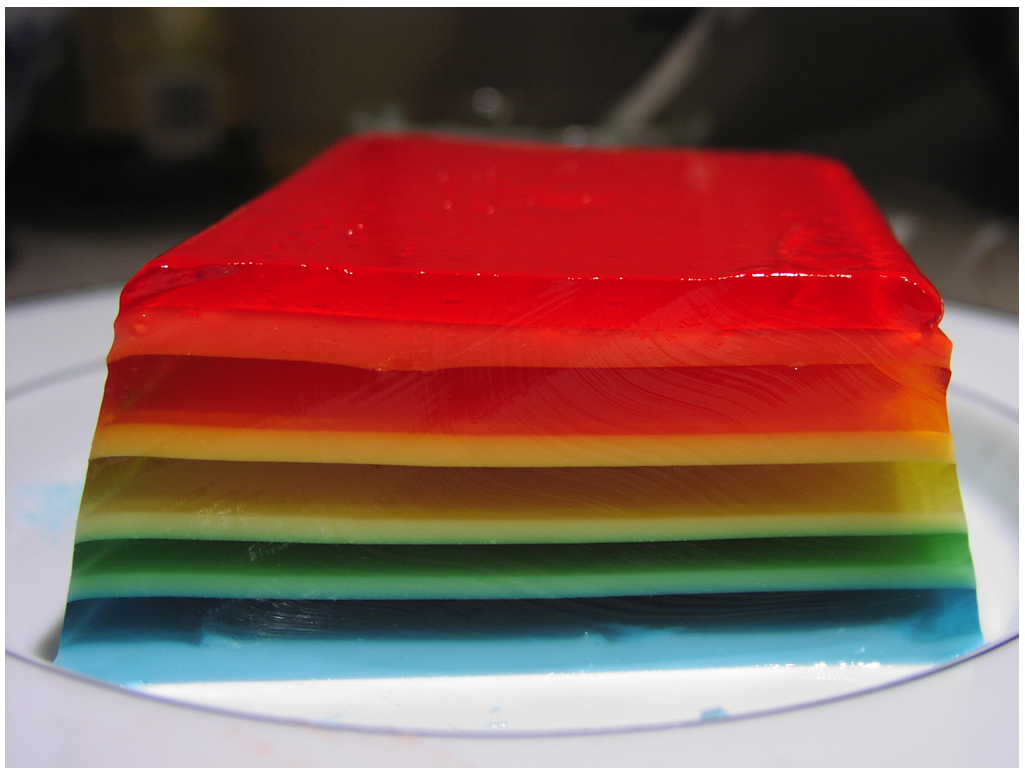
2 Algorithms used

3 Implementation

4 Demo Time!

Demo!

`https://huit.re/mpri-jelly`



CC-BY-SA, Naib @ Wikipedia