6-7-2006

Creating landscapes with simulated colliding plates

Alex Jarocha-Ernst

Creating Landscapes with Simulated Colliding Plates

Alex Jarocha-Ernst

7th June 2006

Abstract

The creation of realistic virtual terrain has been a longstanding computer graphics problem, as terrain will form the backdrop of any virtual world. Approaches to this problem to date have taken one of two approaches: either fractally generating landscapes, or simulating the processes of water and thermal erosion. I have developed a new method to synthesize virtual landscapes, by simulating some of the geological forces that create real-world landscapes I model the collision and deformation of simulated tectonic plates, and create features that mimic those found along real-world plate boundaries. This is achieved through the use of a meshless object representation subjected to physically-based forces, using existing techniques for accurately modeling stress and strain in solid objects.

Master's Thesis Report

Rochester Institute of Technology

Department of Computer Science

Project Website: http://www.cs.rit.edu/~axj7140/thesis

Thesis Comittee:

Chair: Joe Geigel

Reader: Zack Butler

Observer: Warren Carithers

Approved by:

Graduate Coordinator: Name Illegible

Chair: Joe Geigel

Reader: Zach Butler

Observer: Warren R. Carithers

Thesis/Dissertation Author Permission Form

Title of thesis or dissertation: Creating Landscapes With Simulated Colliding Plates

Name of author: Alex Jarocha-Ernst

Degree: Master of Science

Program: Computer Science

College: GCCIS

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, Alex Jarocha-Ernst, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: Alex Jarocha-Ernst Date: 6/8/06

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Inclusion in the RIT Digital Media Library Electronic Thesis & Dissertation (ETD) Archive

I, Alex Jarocha-Ernst, additionally grant to the Rochester Institute of Technology Digital Media Library (RIT DML) the non-exclusive license to archive and provide electronic access to my thesis or dissertation in whole or in part in all forms of media in perpetuity.

I understand that my work, in addition to its bibliographic record and abstract, will be available to the world-wide community of scholars and researchers through the RIT DML. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation. I am aware that the Rochester Institute of Technology does not require registration of copyright for ETDs.

I hereby certify that, if appropriate, I have obtained and attached written permission statements from the owners of each third party copyrighted matter to be included in my thesis or dissertation. I certify that the version I submitted is the same as that approved by my committee.

Signature of Author: Alex Jarocha-Ernst Date: 6/8/06

Contents

List of Figures

List of Tables

1 Introduction

One of the ongoing challenges in computer graphics is the synthesis of realistic artificial landscapes. The rendering of such landscapes is a core component of a realistic virtual world, and methods for designing and synthesizing realistic terrain are required in order to create worlds that are both original and believable. The traditional method of creating such landscapes is through hand-crafted or randomly-generated height fields. More recently, attempts have been made to automate this synthesis by simulating the forces that shape landscapes in the real world. Broadly speaking, these forces can be broken into two groups: erosion and tectonics. Efforts to date, such as [13, 14], have focused exclusively on the former. Erosion by wind or water is, after all, the most visible force shaping the land around us. The landscape features shaped by erosion, however, are first placed there by other forces, most often those generated by tectonic activity. However, there seem to have been no published efforts to simulate plate tectonic action for computer graphics purposes.

I have developed a new method of generating synthetic terrain, based on a model of colliding continental plates. The basic computational mechanisms required to create such a model have already been the subject of much computer graphics research. There are many published techniques for modeling the collision and realistic deformation of objects, and the geologic processes that underlie plate tectonics suggest a way of applying these physically-based techniques to the problem of generating synthetic terrain. In this paper describe the method by which I simulate tectonic plate collisions, and present some results of my simulation.

1.1 Previous Work

Terrain synthesis has traditionally been posed as a matter of generating *height fields*, two-dimensional grayscale images for which high color value corresponds to high altitude in the final rendering. Height fields can be constructed by hand or procedurally, in the latter case usually by pseudorandom noise or fractal-inspired[18] methods.

Only a few attempts have been made to generate terrain by simulating physical processes. Kelley, Malin, and Nielson propose one of the first geologically-based models in [14]; their method models erosion due to stream networks and drainage basins. In [13], Ken Musgrave et al begin with fractal terrain, and apply physically-based models of hydraulic and thermal erosion to produce more naturalistic results than a fractal method alone can achieve. [15] extends [13]'s model of thermal erosion by applying it to a layered data structure instead of the traditional height field, simulating the layers of various rock types that compose the Earth's surface.

Mesh-based physical models of object deformation were pioneered by the authors of [17], and extended to handle a range of deformation types including elastic deformation, plastic deformation, and fracture. A great many methods have been presented in the years since, such as the cloth-rendering techniques presented in [8] and the real-time, Finite Element Method techniques of [6]. Mesh-free methods are a more recent development, beginning with [9, 10]. These methods have an advantage over

mesh-based methods in their ability to accurately simulate the behavior of volumes. Because they contain no explicit connectivity information, mesh-free methods can also handle large deformations more easily than mesh-based ones.

The work presented here draws upon the meshless, point-sample-based methods developed by Pauly, Keiser, Müller, and others in [1, 5, 16]. Their methods are extended in [2] to handle collision of point-based objects, and to realistic modeling of fracturing surfaces in [3].

1.2 Goals

I present a novel, physically-inspired method of building artificial terrain. Computer animation techniques for physically accurate simulation of deforming objects have been a subject of research for many years now, and I demonstrate how those techniques can be applied to the problem of terrain generation. The key lies in the geological process of plate tectonics, which drives the creation of large-scale landscape features in the real world. The system I present can be used as a first step in modeling a physically based, realistic landscape, as it outputs height field images that could then be refined through use of erosion simulations like [13, 14].

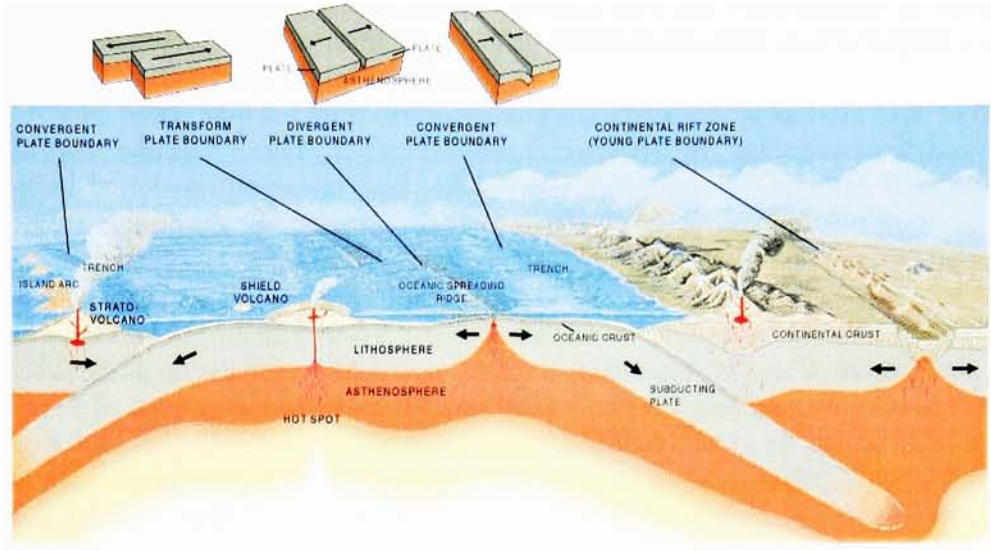
2 Analysis

The method I present here draws on a number of existing geological and physical principles, as well as previous computer graphics techniques. In this section I discuss the underpinnings of the problem of simulating plate tectonics, beginning with a discussion of the theory of plate tectonics itself and proceeding onto the physics of deformation and the components of the meshless object model I have chosen.

2.1 Geological Background

The theory of plate tectonics proposes that the Earth's surface is composed of a number of distinct, rigid rocky plates. These plates "float" on deeper layers of rock with more fluid-like behavior, and are driven to collide with or move away from one another by the internal forces of this fluid layer. They behave in a manner analogous to a fragmented layer of ice atop a pond; each plate is a distinct object that under most circumstances is easier to move as a whole than to deform[11]. For the purposes of this thesis, I am concerned with modeling the behavior of the rigid upper layer, rather than with the source of the forces which drive its motion. The geological structure of the Earth's layers and the various kinds of plate boundaries are illustrated in Figure 1.

Figure 1: Layers of the Earth and plate boundaries[12]



2.1.1 Layers of the Earth

Geologists divide the Earth into several layers, only a few of which are relevant here. The most familiar division is that between the crust and mantle; this division is based on the differing mineral composition of each layer. Alternately, the layers can be divided based on their dynamics; it is this latter kind of division that is of interest in plate tectonics. In this case, the plates themselves are composed of the rigid upper layer, the *lithosphere*, which corresponds to the crust and portions of the upper mantle in the composition model. Below the lithosphere is the more fluid *athenosphere*; while composed of mostly solid rock, the heat and pressure acting on this layer is such that it acts as if it were all of a single, fluid piece[11]. The result of this mechanical division is the ice-on-a-pond analogy mentioned earlier; the rigid plates remain separate and are moved in relation to one another by forces created in the more fluid layer beneath them.

The lithosphere is significantly thicker in some regions than others; specifically, it is thinner in the oceans and much thicker beneath continents and mountain ranges. The pressure and buoyancy provided by the athenosphere is relatively fixed, meaning that wherever a plate's surface rises higher, a greater amount of lithospheric mass is being pushed down into the athenosphere[11].

2.1.2 Plate Interactions

It is the interaction between continental and oceanic plates that produces interesting terrain features. The specifics of these interactions are ruled by a number of physical factors such as plate thickness and density of the composite rocks, but they fall into three broad categories.

Where two plates move parallel to one another, we find *transform faults*. Because of the friction forces inherent in this motion and the rigid nature of their component materials, transform faults build up massive amounts of stress that is suddenly released in the form of earthquakes.

Two plates moving away from one another allow athenospheric rock to rise to the surface, where it quickly cools into new lithospheric matter. The resulting *ridges* are often found on the ocean floor, where the lithosphere is thinnest; the Mid-Atlantic Ridge is perhaps the most famous example.

The collision of plates creates the most visually interesting terrain features, and is the focus of my model. When two plates meet in this manner, the lighter or thinner one is typically forced beneath the thicker plate in a process called *subduction*. Depending on whether each of the plates involved are oceanic or continental, this can result in deep trenches along the subduction zone or the bunching up of lithospheric rock into mountain ranges. The athenospheric rock displaced by the subducted plate is also often pushed to the surface, creating volcanic activity. Subduction activity has created striking features of the landscape in the real world, such as the Marianas Trench, the Himalayan mountains and Tibetan plateau, and the Andes of South America[12]. My work has focused on modelling this kind of interaction.

2.1.3 Driving Forces

The exact driving forces behind plate subduction are still a subject of research, but the process is generally believed to be a combination of several forces including *mantle convection* and *slab pull*[11]. The first results from the fact that the athenosphere is relatively fluid, and hotter the deeper one goes. This produces a convection effect in which hot material bubbles up from below in the center of a region, then cools and begins to fall back down (where it will eventually repeat the cycle). If a plate boundary lies on the edge of one of the *convection cells* formed by this process, the moving athenosphere will tend to carry its lithospheric passenger down with it. Meanwhile, the center of the convection cell likely corresponds to a ridge where new athenospheric rock is being pushed up, which means plates will tend to grow outward from such regions. Slab pull results from a plate that has already begun to subduct; the lithospheric rock composing the sinking plate is much denser than the athenospheric rock surrounding it, and tends to sink. As a result, the remainder of the plate on the surface will be pulled down into the trench as well.

The exact mechanism is not too important for my purposes, but the existence of a downward pull along the plate boundary is. This force will contribute to the formation of tectonic features, and is worth modeling.

2.2 Physical Background

For the purposes of matter on a large scale, it is often convenient to assume that it is truly heterogeneous: we ignore the discrete nature of atoms and molecules, and assume that physical properties such as mass, velocity, and the like are continuously

distributed throughout space. The branch of physics based on this assumption is *continuum mechanics*, and it provides the mathematics that can be used to accurately model deformation in a volumetric object[19]. Properties such as force, velocity, and displacement are modelled as continuous, 3-dimensional vector fields.

2.2.1 Types of Deformation

The deformation of solid objects is divided into three broad categories: *elastic deformation*, *plastic deformation*, and *fracture*. Objects that undergo elastic deformation return to their original shape once the force that caused deformation is removed. On the other hand, materials that deform in a plastic manner retain these changes in shape even after the forces are removed. Objects that are sufficiently deformed will fracture; that is, they will simply break instead of continuing to bend. In real-world objects, there is a smooth transition from each kind of deformation to the next as the amount of force being applied increases.

2.2.2 Strain and Stress

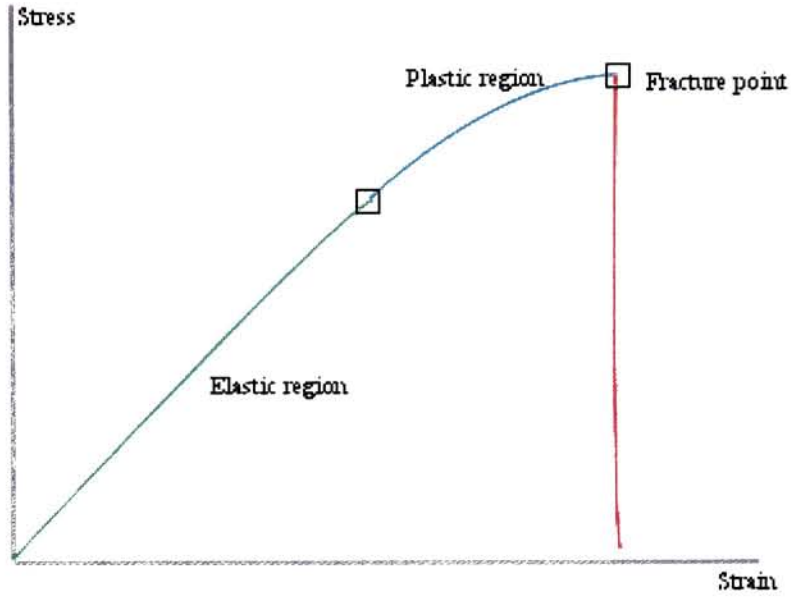
Two physical properties are used to define deformation in a solid object: *stress*(ε) and *strain*(σ). Stresses are forces acting on an object to change its shape; these include whole-body forces like gravity, as well as forces which vary across the volume of an object such as shear and compression force. Strain is the change in an object's shape that results from applied stress. The relationship between these two values is shown by an object's *stress-strain curve* (figure 2). For real-world objects, different degrees of strain have different effects. On a continuum of increasing stress, most objects will deform elastically only up to a certain point. Beyond that, they deform plasticly up to another stress limit, at which point they fracture.

A common simplification is the assumption of a *Hookean material*. Hooke's law states that strain is directly proportional to stress: $\sigma = C\varepsilon$. In the three-dimensional realm continuum mechanics, σ and ε are both rank two tensors, and can be represented by 3x3 matrices. C is a constant that represents the resistance of an object to deformation, and is a rank four tensor; it can be represented as a 6x6 matrix. Although no real material has a truly linear stress-strain curve, Hooke's law is accurate over a wide range of stresses for many materials.

2.2.3 Material Stiffness

Not all materials deform equally in all directions. However, it is convenient to work with an *isotropically elastic* material. Such materials deform equivalently along any axis, and their stiffness tensor C has only two coefficients, E(Young's modulus) and v(Poisson's ratio). In the case of an isotropic material, C can be computed as:

Figure 2: A stress-strain curve.



$$C = \frac{E}{(1+v)(1-2v)} \begin{bmatrix} 1-v & v & v & 0 & 0 & 0 \\ v & 1-v & v & 0 & 0 & 0 \\ v & v & 1-v & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1-2v) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1-2v) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1-2v) \end{bmatrix} \quad (1)$$

In order to make use of Hooke's law, we must also represent the stress and strain tensors as 6-term vectors. We can do so as shown in [19]:

$$\begin{aligned} \sigma_1 &= \sigma_{11} & \sigma_4 &= \sigma_{23} = \sigma_{32} \\ \sigma_2 &= \sigma_{22} & \sigma_5 &= \sigma_{31} = \sigma_{13} \\ \sigma_3 &= \sigma_{33} & \sigma_6 &= \sigma_{12} = \sigma_{21} \end{aligned}$$

$$\begin{aligned} \varepsilon_1 &= \varepsilon_{11} & \varepsilon_4 &= \varepsilon_{23} = \varepsilon_{32} \\ \varepsilon_2 &= \varepsilon_{22} & \varepsilon_5 &= \varepsilon_{31} = \varepsilon_{13} \\ \varepsilon_3 &= \varepsilon_{33} & \varepsilon_6 &= \varepsilon_{12} = \varepsilon_{21} \end{aligned}$$

Using this equivalency with Equation 1, the relationship between stress and strain expands from $\sigma = C\varepsilon$ into:

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} C = \frac{E}{(1+v)(1-2v)} \begin{bmatrix} 1-v & v & v & 0 & 0 & 0 \\ v & 1-v & v & 0 & 0 & 0 \\ v & v & 1-v & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1-2v) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1-2v) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1-2v) \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \\ \varepsilon_6 \end{bmatrix} \tag{2}$$

2.3 Point Cloud Representation

A relatively recent development in the modeling of deformable objects is the use of point-clouds as the underlying object structure. This method discards the traditional mesh, and by doing so avoids many of the difficulties inherent in updating such a mesh in the face of large deformations. It separates the model into volumetric (*phyxels*) and surface (*surfels*) components[1]. The phyxels and surfels that compose an object's representation are effectively point-samples of various continuous object properties.

Phyxels are so named because they model the physical properties of the object; the properties of a phyxel i include position, mass, volume, density, and displacement. This method has been shown to produce realistic flowing and reshaping of objects in [1]. Surfels are maintained to represent the object's surface, and might contain information such as color and surface normals necessary for rendering. In [2] the same authors extend their surfel method to realistically simulate collision and friction forces between two deformable objects, and in [3] they extend it again to model cracking and fractures in the object surface.

I use phyxels very similarly to [1] for the my volume representation, but for reasons of performance take advantage of several simplifying assumptions for my surface representation; these assumptions follow from the model of terrain as a height field, which allows me to ignore parts of the surface that do not face "up". Instead of the surface-based collision methods given in [2], I apply collision forces directly between the phyxels.

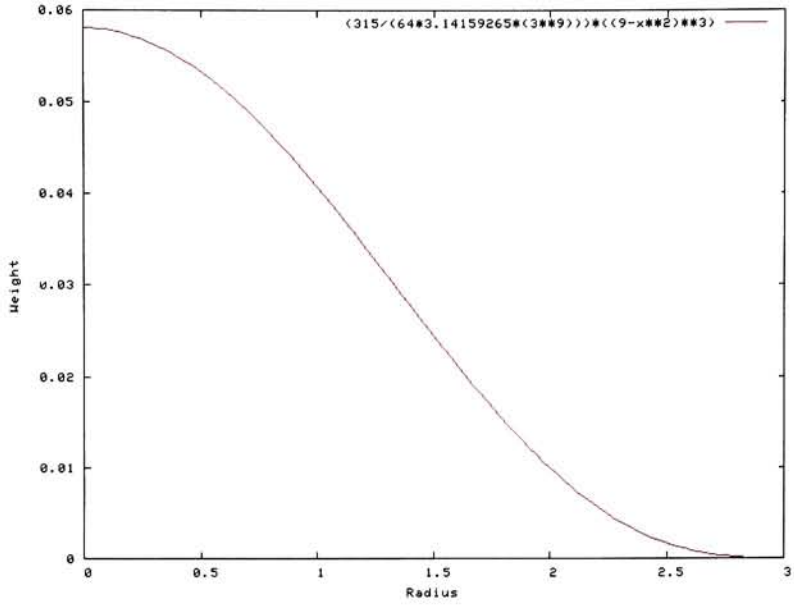
2.3.1 Phyxels

Although each phyxel is only a single point, it actually represents some quantity of matter; it is therefore important that I know the volume and density of that matter. The mass of each phyxel is user-specified; that mass is then distributed around the phyxel point according to a polynomial kernel. I use the normalized kernel function W given in [1] (equation 3).

$$W(r,h) = \left\{ \begin{array}{c} \frac{315}{64\pi h^9}(h^2 - r^2)^3 \text{ if r < h} \\ 0 \text{ otherwise} \end{array} \right\} \tag{3}$$

Figure 3 illustrates the shape of this kernel, using $h = 3$. Given two phyxels i and j, at positions $\mathbf{x}_i$ and $\mathbf{x}_j$, I can use W to calculate the value $w_{ij} = W(\|\mathbf{x}_j - \mathbf{x}_i\|, h_i)$;

Figure 3: The polynomial kernel W



w_{ij} will be used as a weight on several kinds of phyxel interactions. For now, as in [1] the density at phyxel i can be computed as:

$$\rho_i = \sum_j m_j w_{ij} \tag{4}$$

That is, the density at any particular sample point combines the mass contributions of all phyxels whose support radii encompass that point. The volume at phyxel i is then $v_i = m_i/\rho_i$. Although the mass is fixed, the volume and density can change with the object's deformation and should be recalculated after each update for best results (I have found this to have relatively little effect on the final result, so I leave it as an optional parameter for potential performance reasons).

2.3.2 Surfels

A distinct surface representation allows a point-cloud model to contain considerably more detail than is computationally feasible with just phyxels. Each phyxel-based object is initialized with a set of accompanying surfel points, which deform according to the movements of nearby phyxels. Surfels do not have physical properties, but do maintain rendering information such as a surface normal, color, and texture.

I found a simplified surface representation to be sufficient for my purposes, but there are clear advantages in terms of quality to a more complete one, as shown in [1, 2, 3]. Such complete representations require additional computation steps during

each simulation update to maintain an even surface sampling and to preserve detail under large deformation.

2.4 Spatial Hashing

Spatial hashing, described in [7], is a technique used for speeding collision detection in a 3D world. It operates by dividing 3D space into a set of grid cells, each of length l in each dimension. A point's grid location is then hashed into a single subscript. The coordinates $(x, y, z)$of a point are first discretized into grid coordinates (i, j, k): $i = \lfloor x/l \rfloor$, $j = \lfloor y/l \rfloor$, $k = \lfloor z/l \rfloor$. The grid coordinates are passed to a hash function H:

$$H(i, j, k) = (ip_1 \textbf{ xor } jp_2 \textbf{ xor } kp_3) \textbf{ mod } n \tag{5}$$

For equation 5, I set $p_1 = 73856093$, $p_2 = 19349663$, and $p_3 = 83492791$. These may be any large, prime numbers; I opted to use those suggested in [7]. The value n is the size of the hash table. Each entry in the hashtable is actually a bin that may contain a number of point objects (phyxels or surfels), and corresponds to a cell in the 3D grid.

Detecting a possible collision under a spatial hashing scheme begins with determining if the points under comparison hash to the same grid location. If they do not, the two points are considered too distant to have any effect upon one another. If they do, further tests can be applied based on their actual coordinates. This technique allows us to drop distant points from consideration with relatively little computation.

3 Simulation

In this section, I describe the main simulation loop that drives my system. Stress and strain are computed using continuum mechanics equations, and the resulting forces are added to forces for plate motion and plate collision. The resulting displacement of each phyxel then drives the movement of nearby surface elements (surfels).

3.1 Initialization

A number of parameters must be defined by the user before the simulation can start; the details of this initialization process are left for section 4.2. I use two plate objects, one of which is designated the subducting plate; I color the subducting plate red and the other blue in my visualizations, and will refer to them as the red and blue plates respectively for simplicity. The shape of these plates and their boundary is user specified. Each plate is composed of a single material, defined by user-specified elastic constants E and v.

The mass m and support radius h of the phyxels is also user specified, while the volume v and density ρ of each phyxel is then computed from these values. The user specifies the relative motion of the two plates with a single force vector $\mathbf{g}$. Each

phyxel has a force $\mathbf{f}_p$ applied to it to create this motion; for phyxels in the blue plate, $\mathbf{f}_p = \frac{1}{2}\mathbf{g}$, while for phyxels in the red plate $\mathbf{f}_p = -\frac{1}{2}\mathbf{g}$.

3.2 Collision Testing and Response

My collision method is inspired by the surface-based collision forces used in [2], but modified to operate on phyxels directly. Each of the two plates occupies the same set of grid cells, so the first step in detecting a collision between them is to determine, for each such cell, if that cell contains at least one phyxel from each plate. This test is done using spatial hashing, as described above. Collision response operations are then executed on a cell-by-cell basis, only for those cells which contain a potential collision.

My method can be visualized as each phyxel radiating a repelling force that affects only phyxels belonging to other objects. The forces created by any collision are scaled by a spring constant k_s and weighted by w_{ij}; k_s is a simulation constant that must be specified by the user. Higher k_s tends to create more spectacular collisions, but depending on the values of $\mathbf{g}$ and h may cause large crevices to appear between the plates. If k_s is too low, the colliding plates may penetrate one another, which generally causes the simulation to destabilize.

3.2.1 Collision Forces

The collision forces $\mathbf{f}_c$ each phyxel applies to its colliding neighbors are scaled by the dot product of $\mathbf{x}_{ji}$ (the vector connecting the two phyxels under consideration) and the plate force $\mathbf{g}$. The forces are thus maximal for two phyxels being driven directly at one another, and zero for phyxels moving perpendicularly.

For each red phyxel i, I consider all blue phyxels j in its neighborhood and apply a linear repulsive force $\mathbf{f}_{ci}$ from each blue phyxel, scaled by k_s and w_{ij}:

$$\mathbf{f}_{ci} = \sum_j k_s w_{ij} \mathbf{x}_{ji} (\mathbf{x}_{ji} \cdot \mathbf{g}) \tag{6}$$

Similarly, for each blue phyxel j, I consider all red phyxels i in its neighborhood and apply a similar repulsive force $\mathbf{f}_{cj}$:

$$\mathbf{f}_{cj} = \sum_i -k_s w_{ij} \mathbf{x}_{ji} (\mathbf{x}_{ji} \cdot \mathbf{g}) \tag{7}$$

These forces will later be combined with the computed strain forces and certain global forces to determine the change in acceleration of each phyxel for the current time step.

3.2.2 Subduction Force

To simulate the effects of mantle convection and encourage the designated subducting plate to be pulled downward, I apply an additional subducting force $\mathbf{f}_s$ to phyxels in

that plate. This force is each phyxel in the subducting plate that participates in a collision; since it is scaled by w_{ij}, it increases for close colliding phyxels.

$$\mathbf{f}_{si} = \sum_j k_s w_{ij} \begin{bmatrix} 0 \\ -3.75 \\ 0 \end{bmatrix} (\mathbf{x}_{ji} \cdot \mathbf{g}) \tag{8}$$

This aids the formation of trenches along the plate boundary.

3.3 Volume Updates

As in [1], I consider for each phyxel i the influence of all phyxels j in the neighborhood of i. The neighborhood of i contains every phyxel within the same 3D grid cell as i within the support radius h of i, as determined by the spatial hashing technique described in section 2.4. The position vector of phyxel i is denoted as $\mathbf{x}_i$.

3.3.1 Displacement

Using continuum mechanics, we are concerned with the three-dimensional displacement field $\mathbf{u} = (u, v, w)^T$ applied to a volume sampled at coordinates $\mathbf{x} = (x, y, z)^T$. For each of u, v, and w, we will need the spatial derivatives $\bigtriangledown u$, $\bigtriangledown v$, $\bigtriangledown w$ at each sample point in order to compute the strain. The full derivation is shown in [1]; here it will suffice to say that this requires solving three equations of the form:

$$\left(\sum_j \mathbf{x}_{ij} \mathbf{x}_{ij}^T w_{ij} \right) \bigtriangledown u = \sum_j (u_j - u_i) \mathbf{x}_{ij} w_{ij} \tag{9}$$

In the above, $\mathbf{x}_{ij}$ is computed from the positions of phyxels i and j as $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$. The 3x3 matrix $\mathbf{A} = \left(\sum_j \mathbf{x}_{ij} \mathbf{x}_{ij}^T w_{ij} \right)$ is the *moment matrix*; it is fixed at each time step for each pair of phyxels, and having been calculated once can be used to find derivatives for each of u, v, and w. It follows from the above that if $\mathbf{A}$ is nonsingular, for each derivative we have an equation of the form:

$$\bigtriangledown u = \mathbf{A}^{-1} \left(\sum_j (u_j - u_i) \mathbf{x}_{ij} w_{ij} \right) \tag{10}$$

$\mathbf{A}$ is singular if there are less than 4 phyxels in the neighborhood of i (including i itself), or if those phyxels are co-planar or co-linear. This generally results from a too-coarse sampling of the volume, but can also sometimes occur if phyxels are pushed outside the bounds of the volume and into empty grid cells. To avoid the problems caused by a singular $\mathbf{A}$, we can invert $\mathbf{A}$ through use of Singular Value Decomposition as shown in [20]. This method produces results that are close to accurate even in the case of a singular matrix.

3.3.2 Strain and Stress

To calculate strain, I use the Green-Saint-Verant representation shown in [1]:

$$\varepsilon_i = \mathbf{J}_i^T \mathbf{J}_i - \mathbf{I} \tag{11}$$

where $\mathbf{I}$ is the 3x3 identity matrix and the Jacobian $\mathbf{J}_i$ is assembled from the displacement derivatives calculated previously:

$$\mathbf{J}_i = \mathbf{I} + \bigtriangledown \mathbf{u}_i^T = \mathbf{I} + \begin{bmatrix} \bigtriangledown u_i \\ \bigtriangledown v_i \\ \bigtriangledown w_i \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{ui}^T \\ \mathbf{J}_{vi}^T \\ \mathbf{J}_{wi}^T \end{bmatrix} \tag{12}$$

Stress on the phyxel i is then calculated via Hooke's law, as shown in section 2.2:

$$\sigma_i = \mathbf{C}\varepsilon_i \tag{13}$$

3.3.3 Resulting Forces

From the stress term, we calculate two sets of body forces. The first represents elongation, compression, and shear stresses; the second acts to conserve the volume of the deforming object[1].

$$\mathbf{F}_e = -2v_i \mathbf{J}_i \sigma_i \tag{14}$$

$$\mathbf{F}_v = -v_i k_v (|\mathbf{J}| - 1) \begin{bmatrix} (\mathbf{J}_v \times \mathbf{J}_w)^T \\ (\mathbf{J}_w \times \mathbf{J}_u)^T \\ (\mathbf{J}_u \times \mathbf{J}_v)^T \end{bmatrix} \tag{15}$$

The total force applied to phyxel i is then

$$\mathbf{f}_i = (\mathbf{F}_e + \mathbf{F}_v)\mathbf{A}^{-1} \left(-\sum_j \mathbf{x}_{ij} w_{ij} \right) \tag{16}$$

which is the negative sum of the forces applied to each neighboring phyxel j

$$\mathbf{f}_j = (\mathbf{F}_e + \mathbf{F}_v)\mathbf{A}^{-1}(\mathbf{x}_{ij} w_{ij}) \tag{17}$$

Since the forces all sum to 0, these internal forces conserve the linear and angular momentum of the body as a whole.

3.3.4 External Forces

To the body forces on each phyxel, I add several additional forces not specified in [1]. These include the collision forces $\mathbf{f}_c$ and $\mathbf{f}_s$ described in section 3.2, equations 6 and 7, as well as a drag force that acts to stabilize the simulation:

$$\mathbf{f}_{di} = -0.0982 m_i (\text{normalize}(\mathbf{v}_i)) \tag{18}$$

This approximates the action of friction, in that the force is applied in the opposite direction of phyxel i's current velocity $\mathbf{v}_i$. This drag force helps to keep phyxels whose moment matrices become singular from disrupting the simulation.

Finally, the plate-driving force $\mathbf{f}_p$ is applied to each component phyxel of a plate. This force is user-specified, and determines the rate at which the collision will proceed.

3.3.5 Plasticity

Over the course of a time step, a phyxel has two sets of position coordinates, its *reference shape* $\mathbf{x}_i$ and its deformed position $\mathbf{x}_i + \mathbf{u}_i$. Modelling plastic deformation effectively requires that the reference shape at each time step, because otherwise very large strains and elastic forces destabilize the simulation. To this end, each phyxel stores a plastic strain tensor ε_i^p, its *strain state variable*[4, 1], that maintains a record of the strain undergone by this phyxel relative to its initial position in the simulation. The strain considered for elastic forces as described above is then $\tilde{\varepsilon}_i = \varepsilon_i - \varepsilon_i^p$ rather than ε_i. At the end of each time step, plastic deformation is simulated by adjusting the plastic strain and reference position of all phyxels as follows:

$$\varepsilon_i^p := \varepsilon_i^p - \varepsilon_i \tag{19}$$

$$\mathbf{x}_i := \mathbf{x}_i + \mathbf{u}_i \tag{20}$$

$$\mathbf{u}_i := 0 \tag{21}$$

At this point, the phyxel's current velocity and acceleration are updated according to the accumulated forces, and a new displacement $\mathbf{u}_i$ is calculated from these values for use in the next time step. The forces accumulated in this time step are then zeroed before adding those from the next.

3.3.6 State Cleanup

As phyxels move over the course of the simulation, they must be rehashed to ensure that the neighborhood information remains accurate. This is done after updating the plastic state, with a new hash value being compared to the old one. If the hash position has changed, the phyxel is moved to its new neighborhood before proceeding with the next set of calculations.

At this point, I optionally recalculate the density ρ_i and volume v_i of each phyxel as described in section 2.3.1. While this ensures maximum accuracy of the force terms that depend on v_i, doing so after every time step produces a not-inconsiderable slowdown, and seems to have relatively little in the way of visible results.

3.4 Surface Updates

The surface representation of my plate objects is literally carried along by the volume representation. Compared to [1, 2], on which this work is in large part based, I use a relatively simple surface model; I briefly note differences between my representation and that used elsewhere.

3.4.1 Displacement

As with the phyxels, I apply a displacement field $\mathbf{u}_{sfl}$ to a set of position coordinates $\mathbf{x}_{sfl}$ representing the current state of the surfels. The vector field $\mathbf{u}_{sfl}$ is computed from the displacement field $\mathbf{u}_i$ applied to each neighboring phyxel i in the current time step; for this, I reuse the the value of $\nabla\mathbf{u}$ computed via equations 9 and 10. In order to avoid tears in the surface, all phyxels whose support radius h encompasses this surfel is considered neighboring for the purpose of this computation, regardless of grid cell location. For each surfel, the displacement vector is calculated as:

$$\mathbf{u}_{sfl} = \frac{1}{\sum_i W(\mathbf{x}_{sfl} - \mathbf{x}_i, h_i)} \sum_i W(\mathbf{x}_{sfl} - \mathbf{x}_i, h_i)(\mathbf{u}_i + \nabla\mathbf{u}_i^T(\mathbf{x}_{sfl} - \mathbf{x}_i) \qquad (22)$$

This is similar to displacement approach described in [1], except that I reuse W as my weighting function in place of their Gaussian weighting function. $\mathbf{u}_{sfl}$ is added to the surfel's position; although I ignore normal information in my simulation, a more general surface animation scheme would apply $\mathbf{u}_{sfl}$ to the surface normals as well, as described in [1, 16].

3.4.2 State Cleanup

As with phyxels, surfels may need to be rehashed after each time step to ensure proper neighborhood tests.

In more general point-based models, additional steps are usually applied to maintain an evenly-spaced surface sampling. These methods often include some kind of split-merge scheme to ensure even sampling, followed by projection onto an implicit surface to maintain fine details; see [1, 2, 5, 16]. For my purposes, the initial sampling is generally sufficient, and adding resampling operators requires too much of an increase in complexity and computation time for the benefit it gives.

4 Implementation

In this section I provide details of my implementation that go beyond the simulation loop described above, including the software structure, external libraries used, and the initialization and rendering procedures.

4.1 Component Overview

The software used for this paper is a combination of custom-built C++ code and several externally developed open-source libraries.

4.1.1 Software Components

The major software components of my system are:

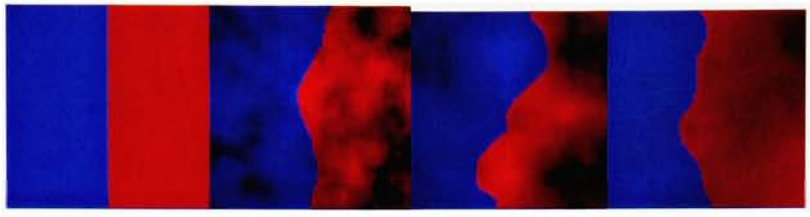
- *tectonic*: The main program, containing initialization code and the collision operations.
- Class *PhyxelObject*: Used to represent a single plate; contains spatial hashing and strain/stress calculation code.
- Class *Phyxel*: Data container for phyxel properties, as well as basic update code.
- Class *Surfel*: Data container for surfel properties, as well as surface update code.
- Class *GLPhyxelRenderer*: Used to give live, visual feedback on the simulation's progress. See section 4.3.1.
- Class *HeightfieldWriter*: Used to save the surface representation as a height field for external rendering. See section 4.3.2.
- Class *PlateBuilder*: Used to generate initial plate shapes. See section 4.2.1 for details.

4.1.2 Supporting Libraries and Tools

I used a couple of external tools to create the output shown in this paper, and a number of other libraries to perform mathematical and image operations.

- The freeware ray-tracer POV-ray[21] was used to render the height fields created by my software.
- The ImageMagick[22] suite of image editing tools and its C++ API, Magick++, was used to perform image input, output, and drawing operations.
- libgfx[24] was used for general-purpose vector and matrix math.
- The Gnu Scientific Library[23] was used to perform the SVD operations needed to safely invert $\mathbf{A}$.

Figure 4: Several 80x80 input images



4.2 Initialization

The initial state of the system is specified by a combination of command-line flags and an input image that describes the starting landscape. All the command line flags have default values, which are used if no others are specified.

4.2.1 Starting Landscape

The system's initial state is read from an input image (several examples of which are shown in Figure 4), which defines the shape of the plates that will be colliding. The input image defines both the shape of the plate boundary, and that of the plate surface. It consists of a blue and red region, corresponding directly to the surface of the "blue" and "red" plates. For the blue region, the value of a pixel's blue color channel determines the height of a corresponding surfel, while the value of a pixel's red color channel determines the height of the surfels in the red region. A pixel cannot belong to more than one region; the larger of the red and blue channels determines to which it is assigned.

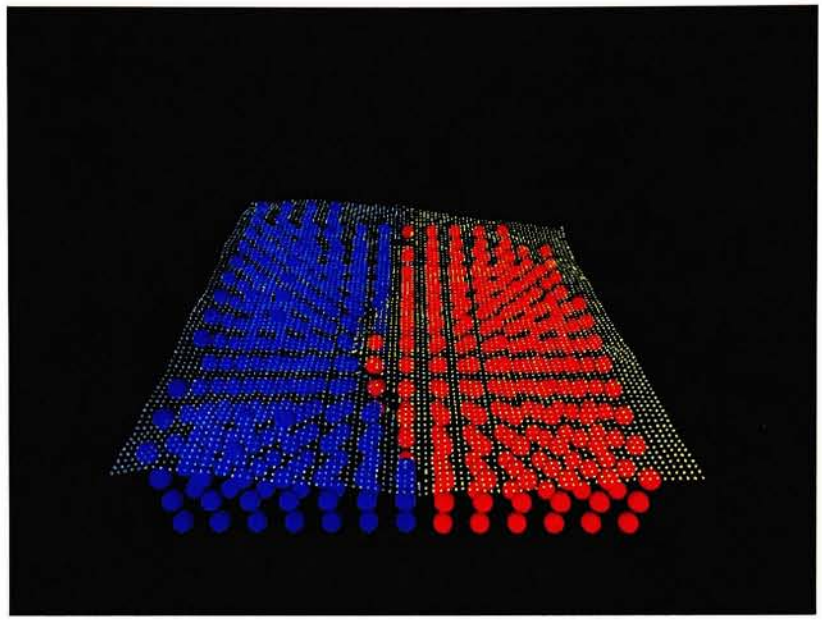
Phyxels are distributed in an evenly-spaced grid beneath the surface representation, and assigned to one plate or another according to the assignment of the surface directly above them. The thickness of the phyxel volume ranges from 2 to 4 phyxels according to the height of the surface. See Figure 5 for an example of this construction.

Each phyxel and surfel is hashed to a grid location and corresponding bin in its container object as it is created. Optionally, the system can be run with no surface representation; this produces faster updates while still showing the interaction of phyxels, but the lack of a surface representation prevents the system from creating height field output.

4.2.2 Physical Parameters

A number of user-specified parameters are used to specify the physical properties of the simulated plates. The parenthetical notation after each parameter notes whether it is stored on the phyxel or object level. A brief description of its meaning and its default value follows each parameter.

Figure 5: Phyxels and surfels constructed from an uneven input image



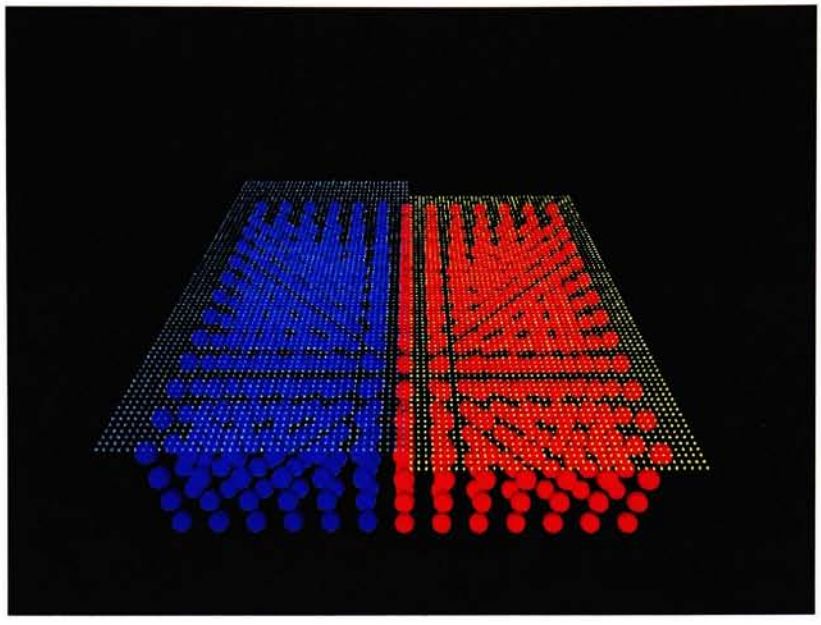
- Phyxel mass m: The mass each phyxel will be initialized with. Defaults to 12000, giving a set of densities approximately that of granite.
- Young's modulus E: Elasticity constant; will be used to compute the stiffness matrix $\mathbf{C}$. Defaults to 45, an average value for natural stone.
- Poisson's ratio v: Elasticity constant; will be used to compute the stiffness matrix $\mathbf{C}$. Defaults to 0.25, and average value for natural stone.
- Collision vector: Specifies direction and magnitude of the collision force; this vector will be scaled by the spring constant k_s to produce the actual collision force $\mathbf{g}$.

4.2.3 Simulation Parameters

Another set of parameters have no equivalent physical property, but determine aspects of the simulation's behavior. Here I briefly describe the meaning of each parameter and give its default value. Often these must be tweaked to get the best possible results from a particular input image.

- Grid size: The size of the grid cells used for spatial hashing and neighborhood calculations. Defaults to 13.
- Hashtable size: I use a neighborhood hashtable consisting of 125 bins.

Figure 6: Initial state of a mostly-flat landscape



- Support radius h: Radius across which each phyxel's mass is distributed. Defaults to 6.
- Spring constant k_s: Multiplier applied to collision force vectors. Defaults to 15000000.

4.3 Rendering

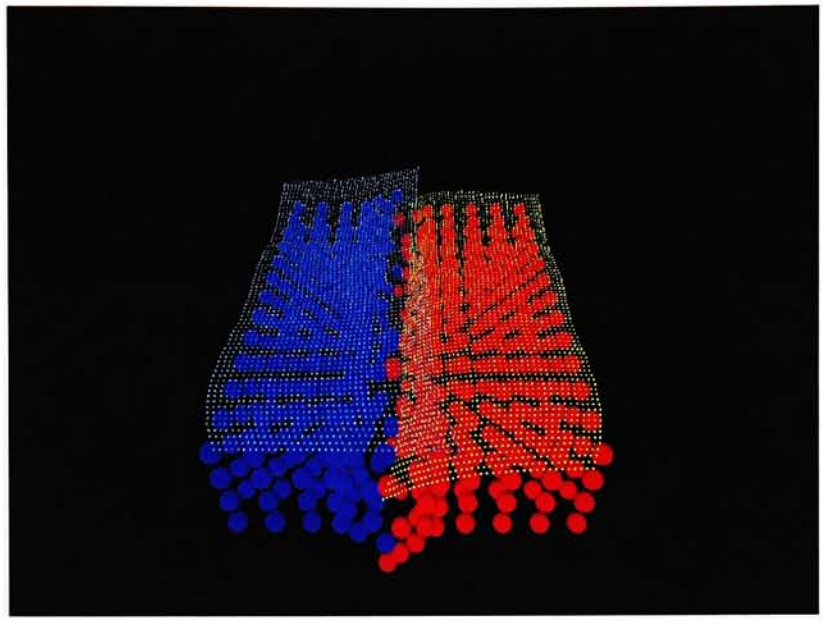
I have implemented two rendering schemes for the terrain generated by my system. The first uses OpenGL to provide a real-time view into the plate collision, enabling the user to stop and start the process at will as interesting stages are reached. The second is a height field-export scheme that creates a grayscale image of the sort used by many external programs to render terrain objects.

4.3.1 OpenGL Live Rendering

The live rendering scheme is simple enough that its use does not significantly impact the computation time of each simulation step. It renders each phyxel as a sphere, colored according its object membership, while the surfels are shown as much smaller spheres. Figures 6 and 7 show such renderings.

Alternately, this scheme can be set to color the phyxels according to their hash value, allowing the user to examine the boundaries of the spatial hashing grid this

Figure 7: The flat landscape after 200 time steps



effect is demonstrated in figure 8. The user can also start and stop the simulation, or save the current terrain object as a height field image.

4.3.2 Height Field Rendering

I provide functionality to save the current state of the terrain object as a grayscale height field, suitable for rendering in many external programs. The output of this method is grayscale image in which pixels with high value correspond to surfels with high y positions.

When the save command is issued, the height field image is initialized to all black. For each surfel in the simulation, I convert its x and z position values to pixel (x, y) coordinates in the image, and draw a circular patch of color whose value is scaled according to the surfel's height. In cases where multiple surfels may cover the same image region, I draw only the highest; this generally occurs when portions of the subducting plate's surface is pulled beneath that of the overlapping plate. Over the course of this process, I track the lowest surfel value encountered so far; once all surfels have been draw, I set any remaining black pixels to this value to produce a smoother output image.

Height field output created from the landscapes in figures 6 and 7 is shown in figures 9 and 10, repspectively.

Figure 8: Phyxels colored by hash value.

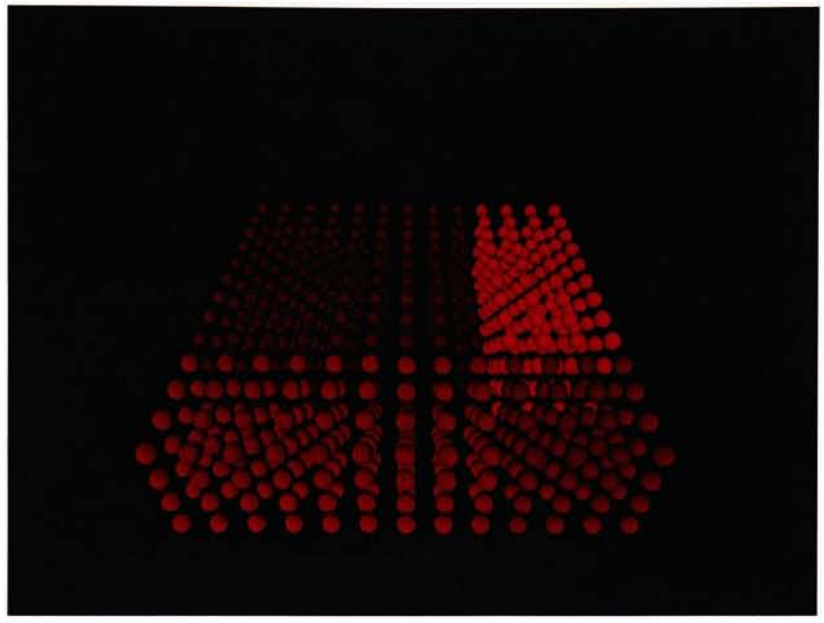


Figure 9: Height field output from the mostly-flat landscape

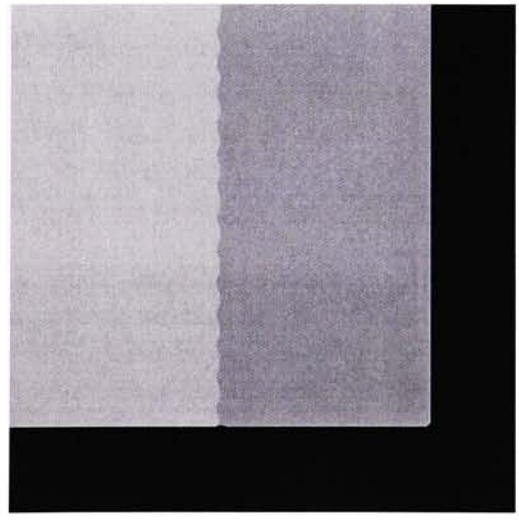
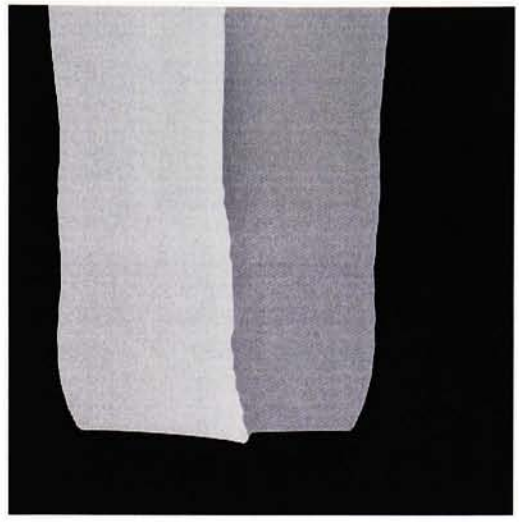


Figure 10: Height field created after 200 time steps.



5 Results

In this section I present and discuss the results of a number of experiments performed with the system I have detailed above. For each experiment, I show a sequence of images showing a height field created at 0, 50, 100, 150, and 200 time steps. Beneath each height field is a 3-dimensional rendering of the same, created with POV-ray.

Unless otherwise noted, the various physical and system parameters use the defaults given above.

5.1 Simple Collisions

In order to test my system's effectiveness at the most basic level, I've run a number of experiments with a relatively flat landscape (figure 11). The right-hand, subducting plate is started at a slightly lower height than the left hand one, to encourage the subduction effect. The plate boundary is flat, and perpendicular to the collision vector g.

The progression of a basic, head-on collision is shown in figure 12. This result demonstrates several important effects, all of the form we should expect from our knowledge of real-world subduction boundaries. We can see that the subducting plate is pulled downward as it approaches the boundary, creating a deepening trench as the simulation progresses. Likewise, the subducting plate is compressed by the collision, and forms a hill region just beyond the trench. Finally, the top plate is pushed upward along the boundary to form a distinctive ridge or mountainous region.

An angled collision (figures 13, 14) produces similar results, but with the added deformation of the plates beginning to deform around one another once they have

Figure 11: Flat input image

room to do so. This appears physically plausible, but does not have a direct correspondence in real-world geology, in which the many continental and oceanic plates are tightly interlocked.

Reducing the support radius(figures 15, 16, and 17) compresses the range at which the collision force acts, so we do not see as pronounced an effect along the plate boundary. This also seems to produce "grainier" results, with the surface deformation being less smooth. This effect is not too surprising, as each surfel will be subject to the influence of fewer phyxels in this case.

5.2 Colliding Existing Landscapes

I then constructed a series of randomly-generated landscapes with hand-drawn uneven boundaries(figure 17), to test the system's ability to transform uneven terrain in a physically plausible manner. Similar results are obtained, as shown in figures 18-20, with the edge of the top plate along the boundary being pushed upward as the simulation progresses. Trench formation is not as visible in these images, however. The uneven boundary actually remains very similar over the course of the simulation, although regions where the top plate pushes deep into the subducting plate often seem to undergo more visible rise.

5.3 Varying Physical Parameters

As they are composite materials, natural stones tend to have widely varying density and elastic constants across samples. The default mass (12000) was chosen to produces densities approximating that of granite (which has an average density of around 2700 kg/m^3). By contrast, setting the mass to 8000 approximates a lightweight limestone with density of 1800 kg/m^3. I tested progressions of a head-on collision with a less massive subudcting plate(figure 21) or less massive upper plate(figure 22). The results are what one would expect: the lighter plate deforms more easily under equivalent forces. The effect of reducing the top simulated plate's mass is most notable, as it causes a visibly increased ridge height along the boundary.

Likewise, I tested the effects of varying the plates' modulus of elasticity across a range common for stones such as limestone and granite. The values of 70 and

Figure 12: Head-on collision of flat landscape with default parameters

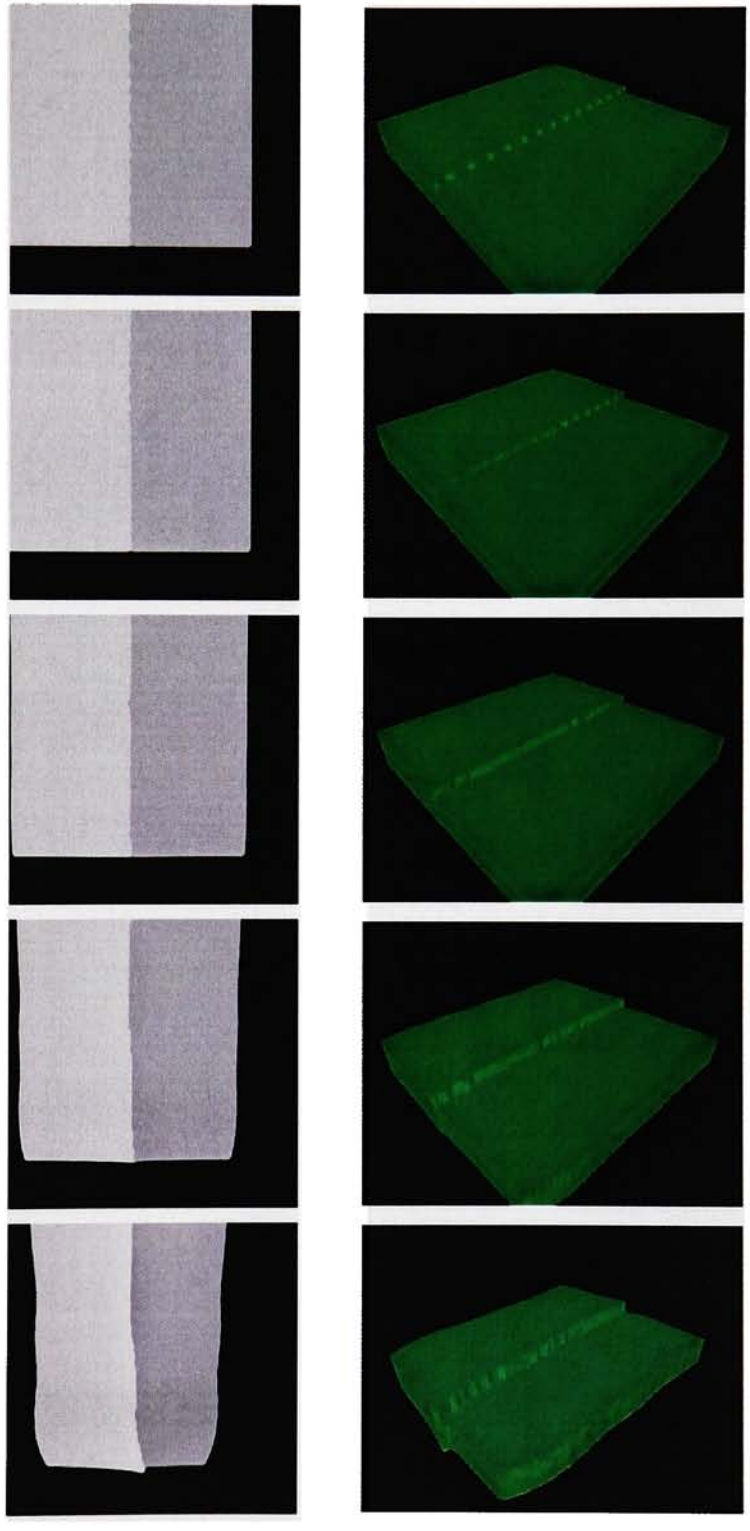


Figure 13: Collisions at a $-45°$ degree angle.

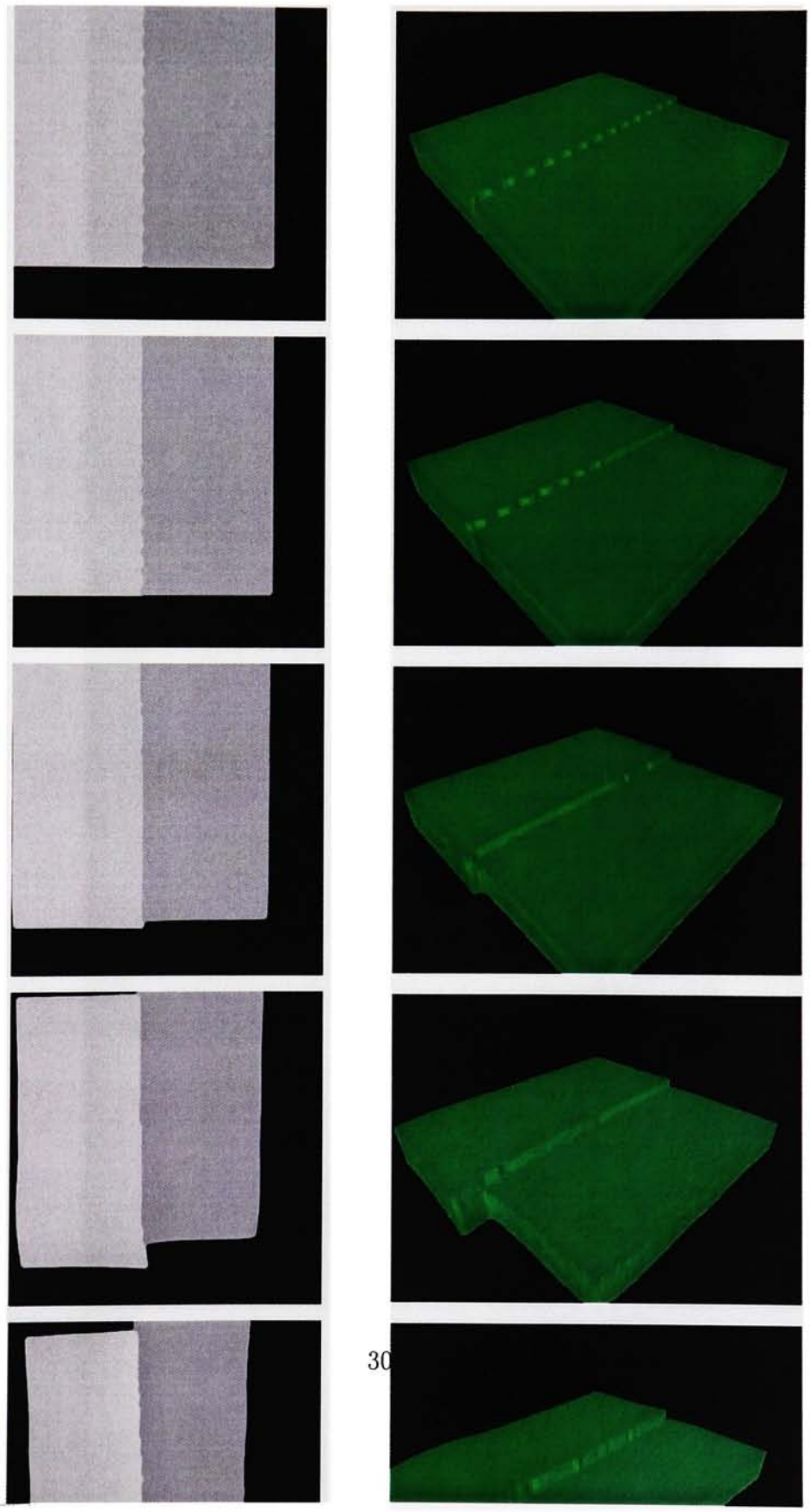


Figure 14: Collisions at a 45° degree angle.

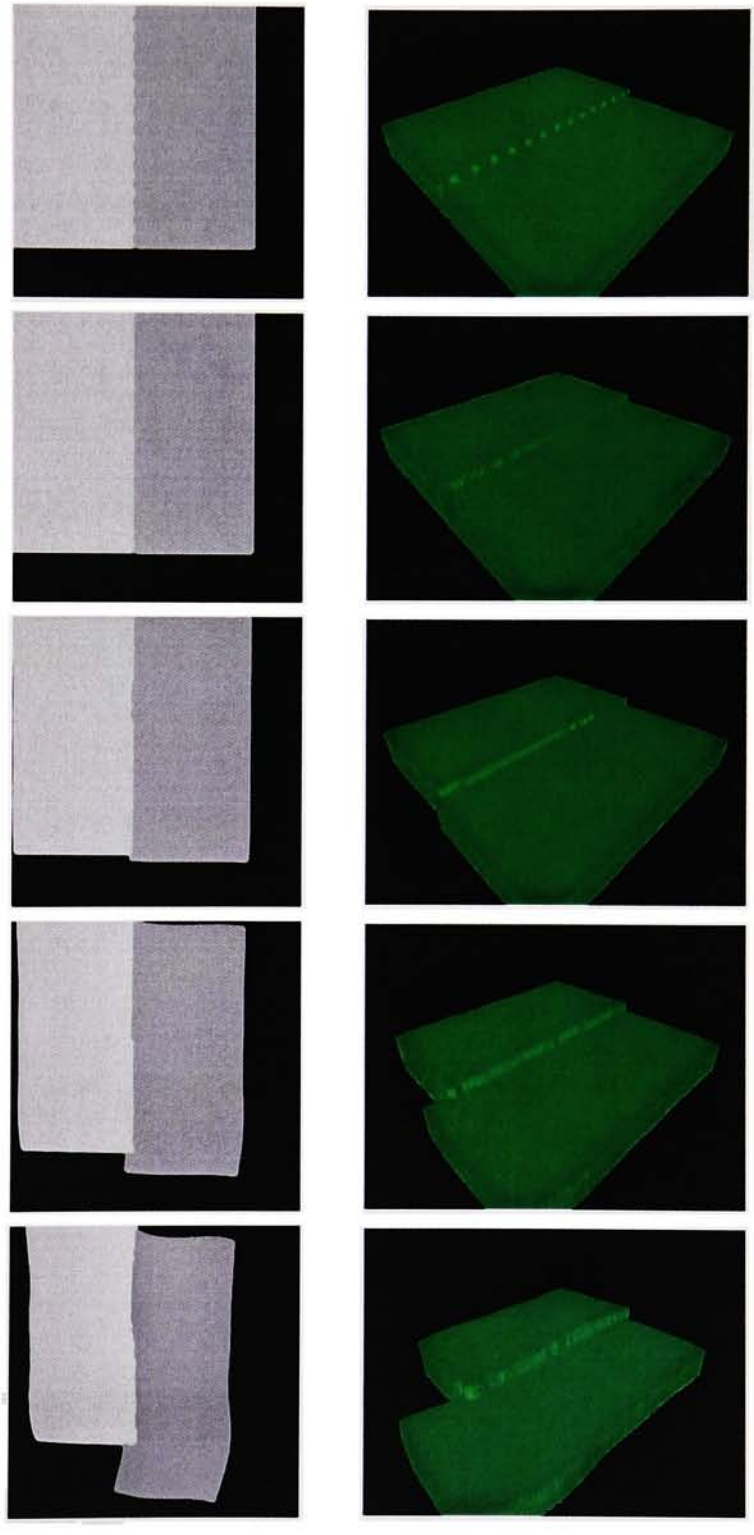


Figure 15: Head-on collision with $h = 3$

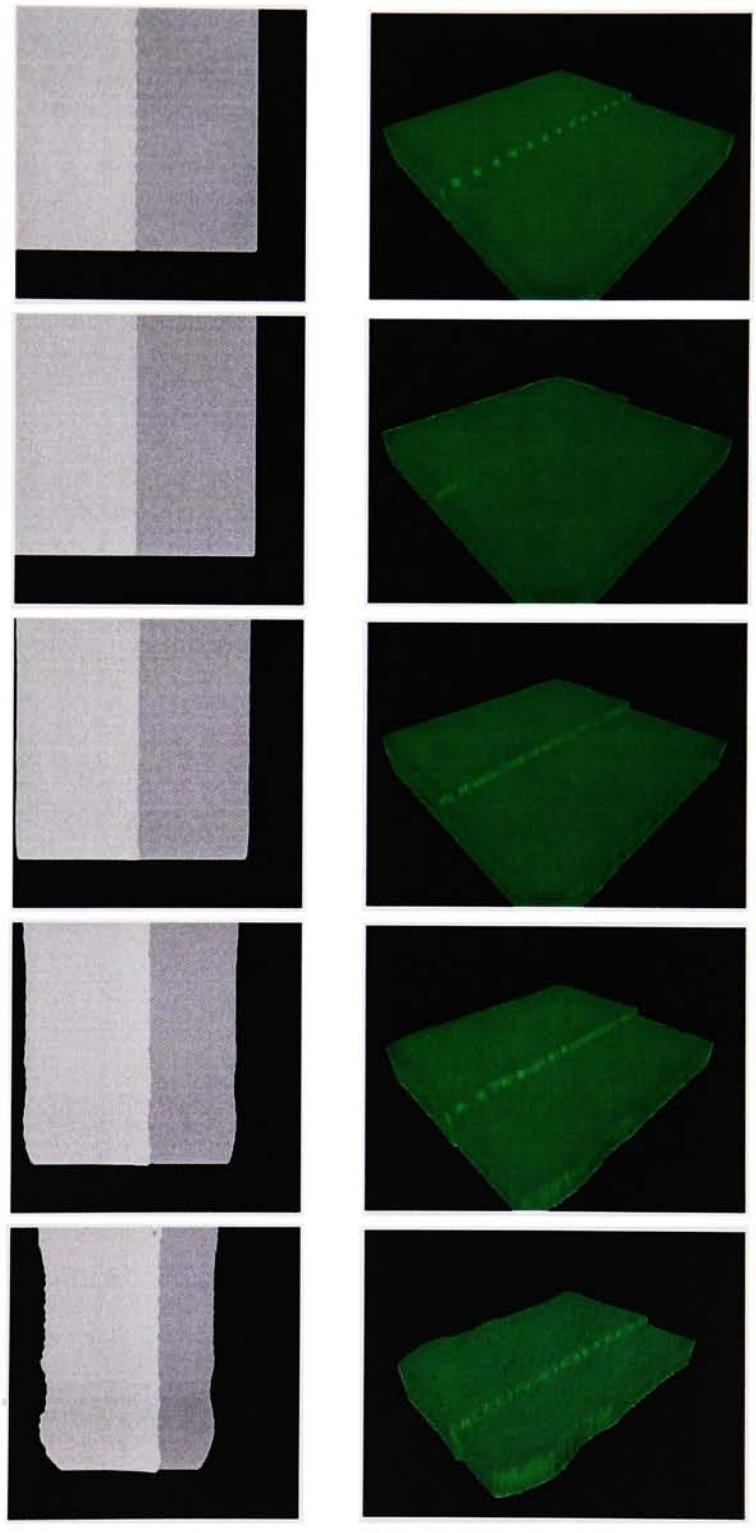


Figure 16: Angled collisions with $h = 3$

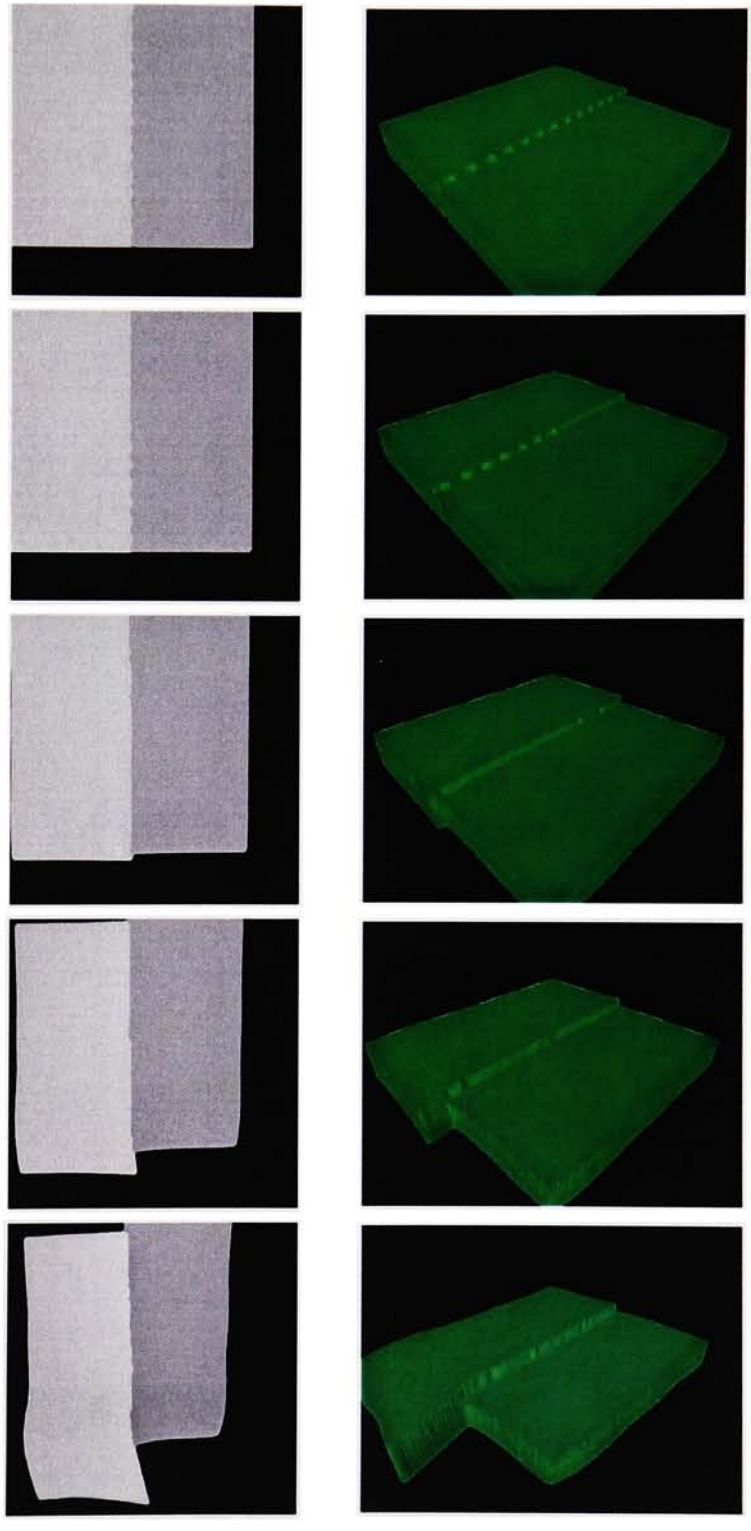


Figure 17: Opposite angled collision with $h = 3$

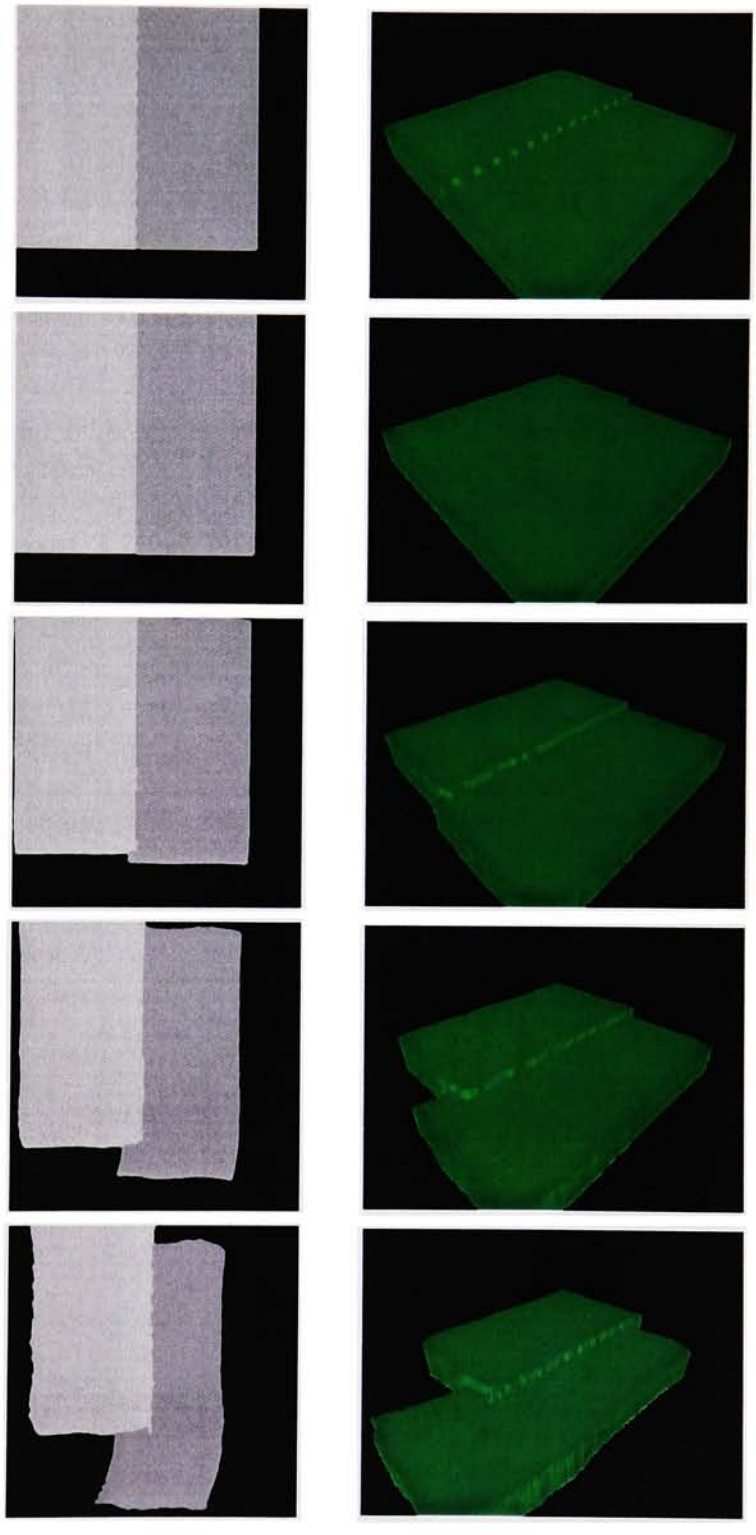
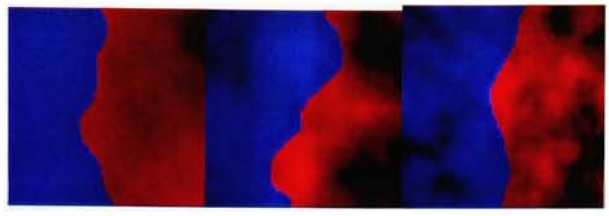


Figure 18: "Shore", "Mountain", and "Island" input images



20 are approximately the upper and lower limits found in nature for these types of stone. The results are shown in figures 23 and 24. As may be seen there, I found that changing the moduli within these limits did not produce any particularly visible results; it takes a Young's modulus well outside the normal range for stone to produce a notable change in the plate deformation.

5.4 Execution Time

To examine the execution time in detail, I recorded the time needed for the system to advance 100 time steps in batch mode and render one height field image. The OpenGL renderer is relatively lightweight, and its render time is insignificant in comparison to the calculation time. For comparison, I show the time used to calculate those 100 steps with and without surfels, with and without density recalculation, and over several grid cell sizes (tables 1 and 2). All runs used the flat landscape, which had about 600 phyxels and 6400 surfels.

Table 1: Results on a SunBlade 1500 workstation with 1GB of RAM, running Solaris 9

	cell size = 13	cell size = 9	cell size= 7	cell size = 5
No surfels	2m 20s	1m 38s	1m 6s	38s
+ density recalculation	2m 37s	1m 50s	1m 22s	43s
Using surfels	10m 26s	9m 26s	9m 50s	9m 6s
Surfels + density	10m 25s	9m 30s	9m 35s	9m 44s

Table 2: Results on an Athlon 2000XP PC with 1GB of RAM, running Gentoo Linux.

	cell size = 13	cell size = 9	cell size= 7	cell size = 5
No surfels	44.6s	31.3s	22.1s	12.7s
+ density recalculation	43.4s	36.5s	24.1s	14.4s
Using surfels	3m 17s	3m 12s	3m 1.6s	2m 59s
Surfels + density	3m 21s	3m 16s	3m 3.9s	2m 59s

Figure 19: Head-on collisions for "shore" landscape.

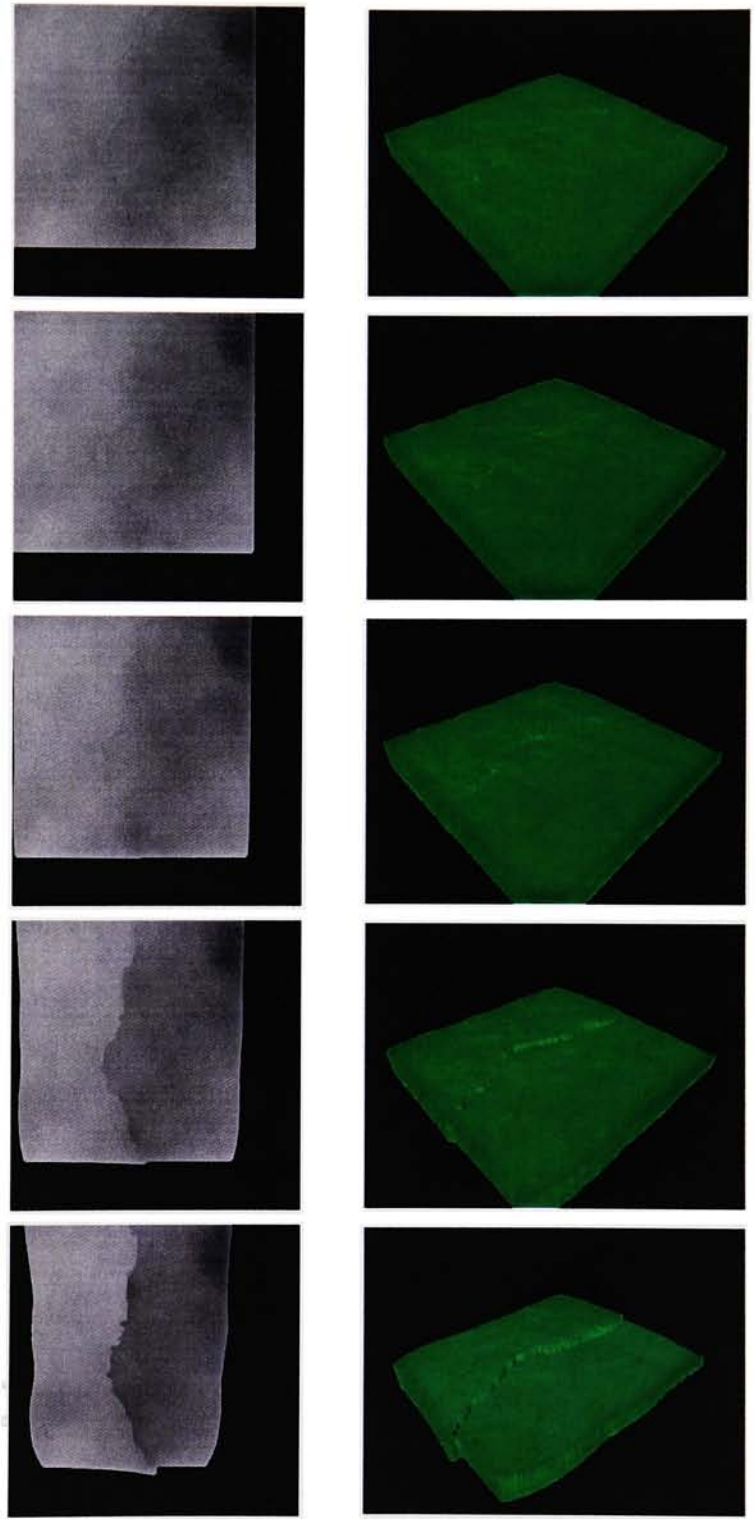


Figure 20: Head on collision using "mountain" landscape

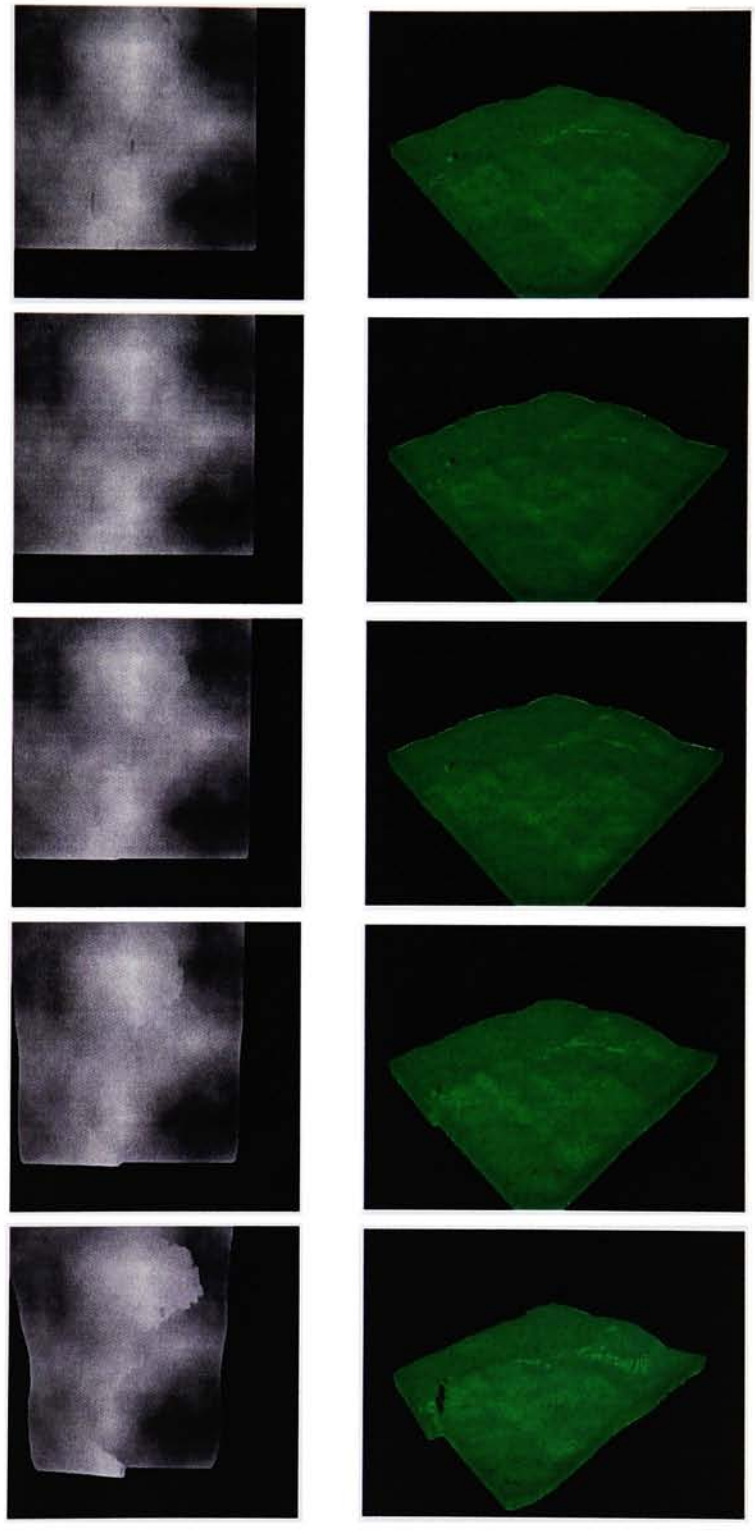


Figure 21: Head on collision using "island" landscape

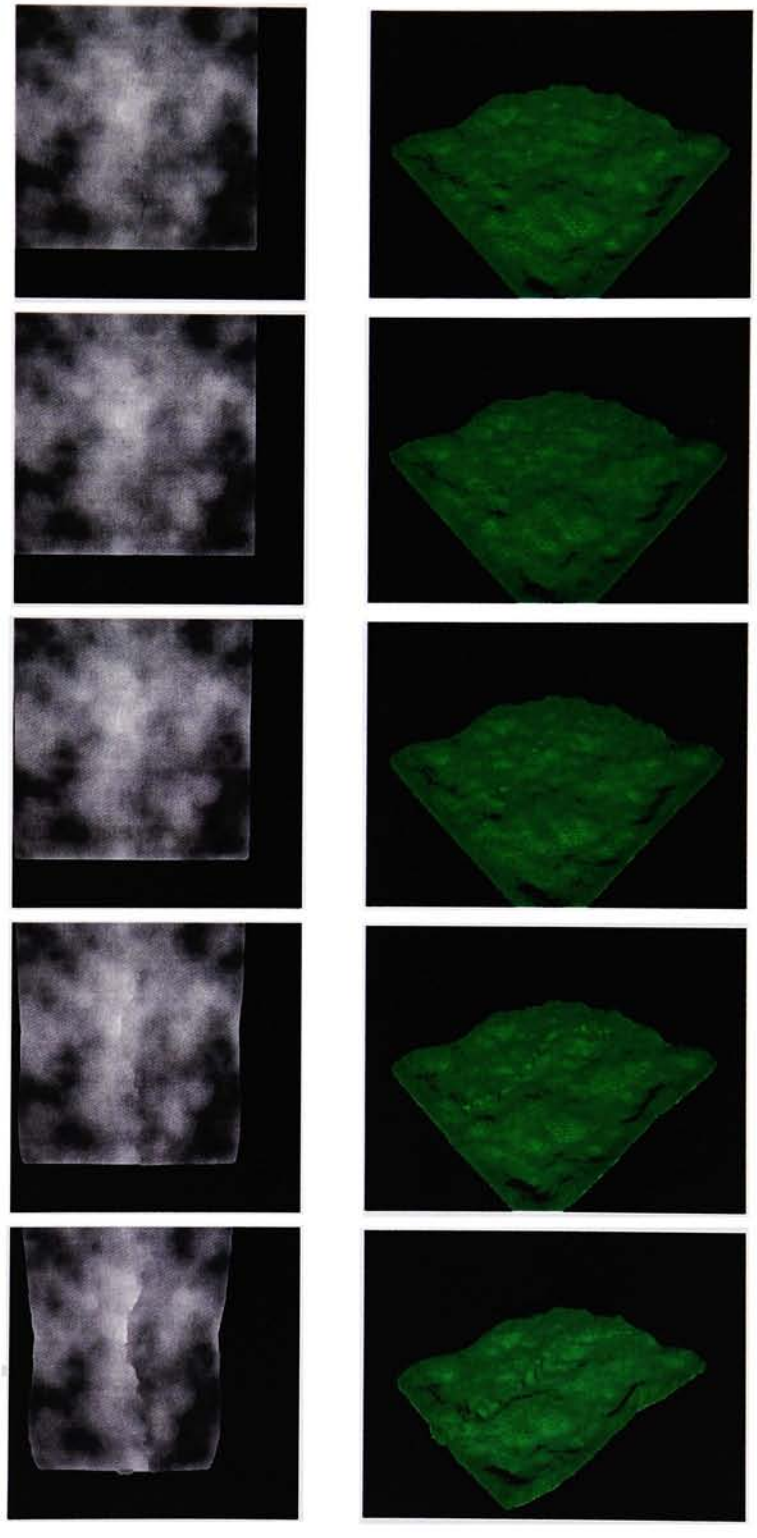


Figure 22: Collision with a less massive subducting plate ($rm = 8000$)

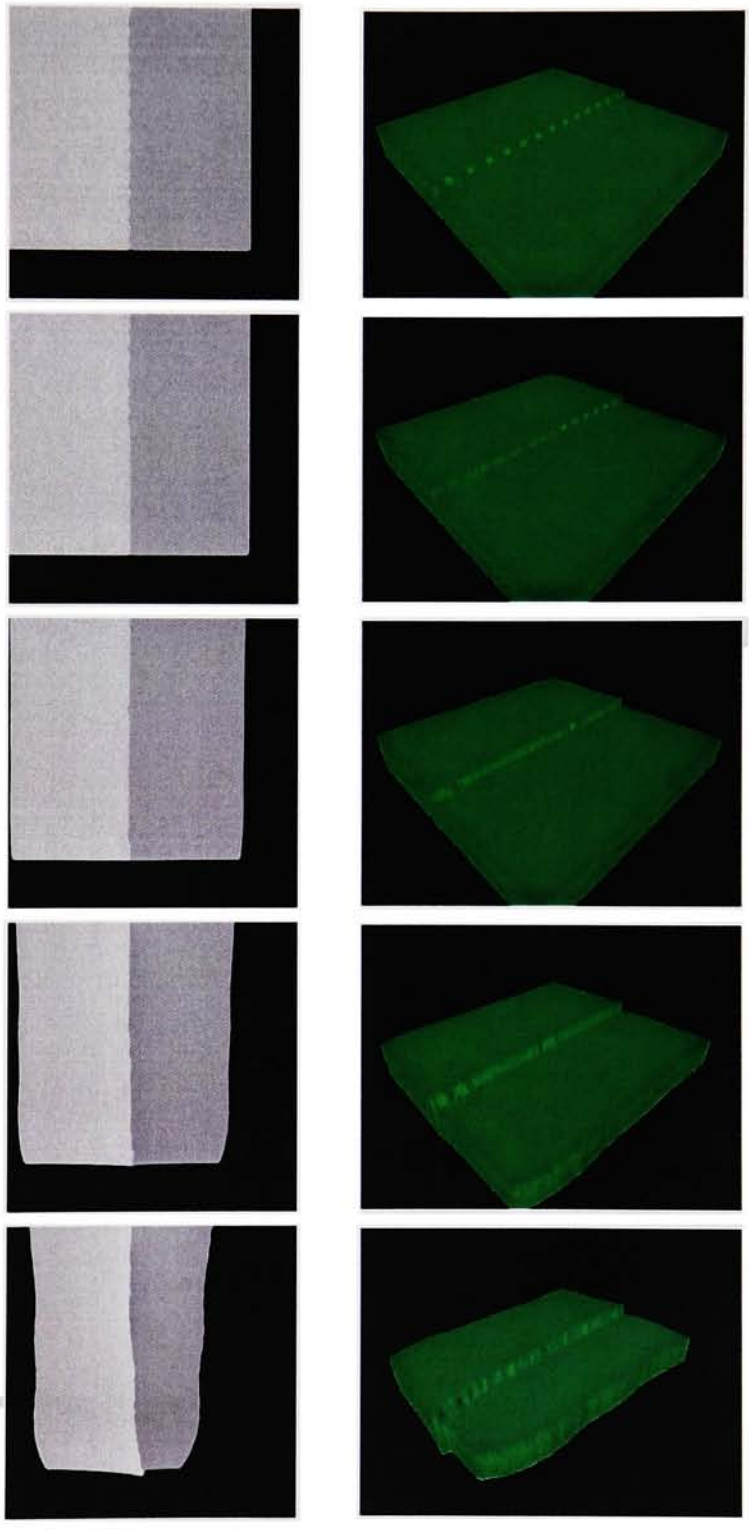


Figure 23: Collision with a less massive top plate ($bm = 8000$)

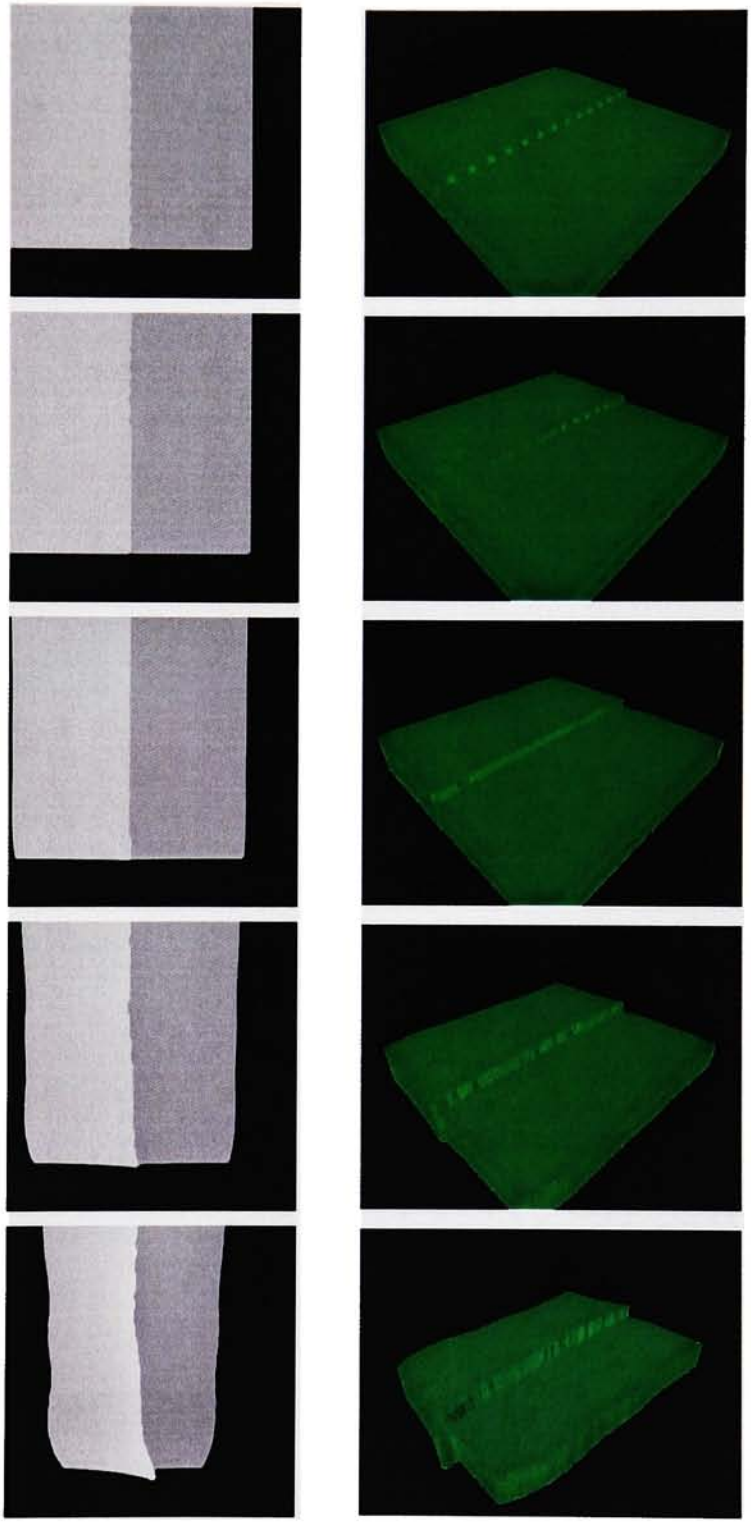


Figure 24: Collision with $by = 70$, $ry = 20$

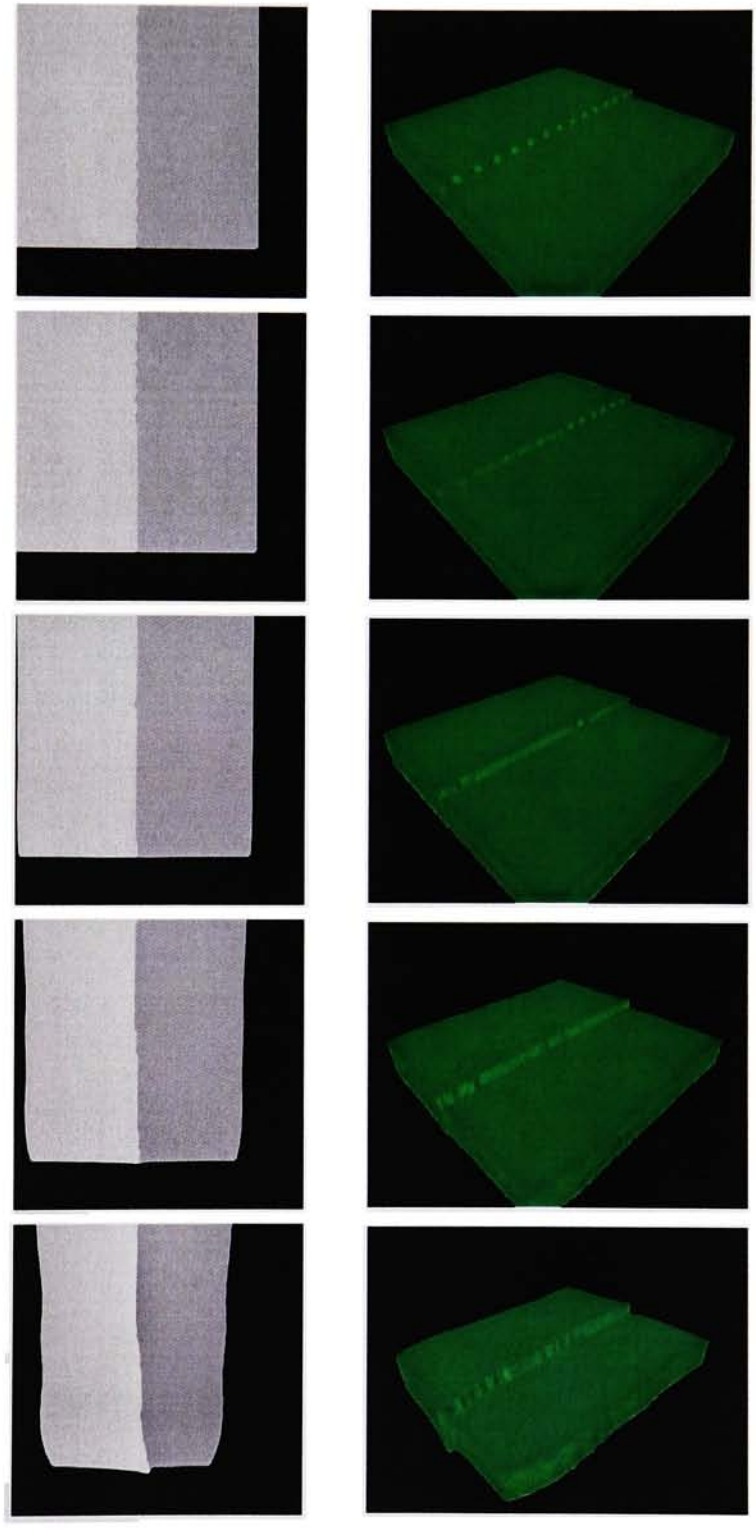
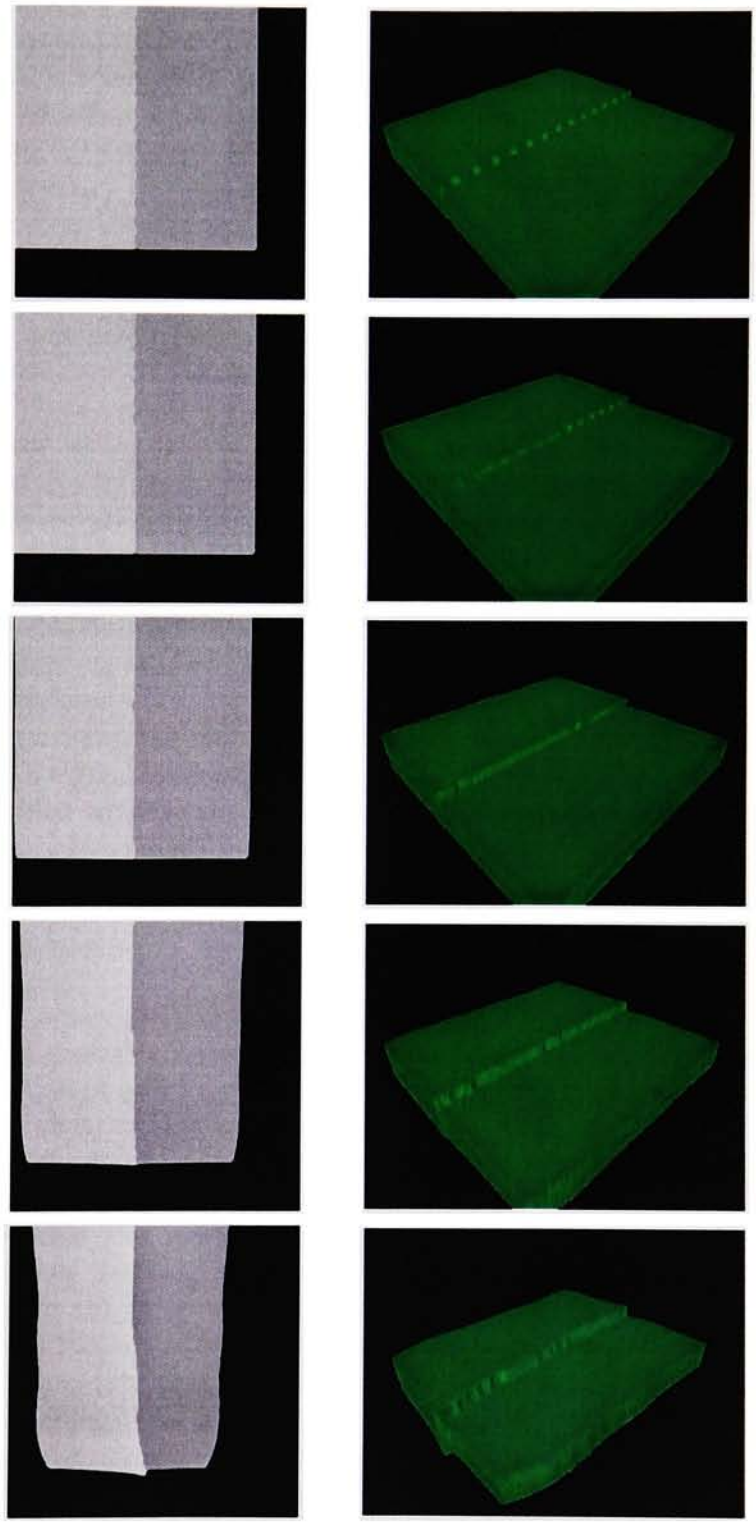


Figure 25: Collision with $ry = 20$, $by = 70$



Although reducing the grid size can provide an appreciable speedup, it often has drastic consequences for the quality of the resulting terrain. Small grids are more likely to contain only a few phyxels, and increase the frequency of the discontinuities that appear along grid edges. This is particularly evident when grid edges align with the plate boundary, which tends to result in a very unstable simulation. The amount of time used by updating the surface representation does not change with the grid cell size, as surfels do not consider only phyxels in the same cell. This is an unfortunate trade-off, but necessary to eliminate surface tearing along the grid boundaries in my simplified surface representation.

Even with my simpler surface model, I have fallen far short of the performance described in [1], which uses a much more complex, multi-stage surface representation to ensure even sampling. There, speeds of 27 fps are achieved with smaller (200) numbers of phyxels and 10,000 surfels; this decreases to 12 fps for 400 phyxels and 20,000 surfels. These results are achieved on a fairly typical laptop computer. The greatest difference seems to result from their surface implementation; I made an abortive attempt to implement the surface resampling method as is described in [1, 5], and even in early stages found the time per frame rising as high as 30 seconds with ~10,000 surfels, even when using spatial hashing for neighborhood tests. This vast difference suggests that their method is greatly optimized in some fashion, but their reports do not delve into that level of detail. Given the large number of calculations (particularly matrix operations) performed at each step, a more efficient math library is one likely candidate.

While not real-time, the simulation is speedy enough that the user can watch it progress without great difficulty. I have often found it easier to test a run with the surface disabled to view the basic form of the results, and enable the surface representation only to produce more permanent output.

6 Conclusions

I have developed a new method of creating computer-generated terrain objects that is physically- and geologically-based. The existence of techniques for simulating the realistic deformation of colliding objects allows us to pursue this new kind of terrain generation, by simulating the natural processes of plate tectonics. My work has focused on colliding plates and simulating plate subduction and the plate-boundary features that result from it.

I have shown that this method is capable of realistically approximating certain large-scale landscape features common to subducting plate boundaries, including mountain ranges or ridges on the upper plate and trenches formed where the subducting plate is pulled into the athenosphere. However, my method suffers from a problem of scale; creating properly detailed terrain would require a much larger or more detailed sampling than is computationally feasible with the techniques shown here. It does, however, provide approximate height fields that can be used with other physically-based terrain generation techniques such as those based on erosion [13, 14].

6.1 Future Work

The authors of [1, 2, 3] have used their phyxel and surfel model to quickly simulate collisions between detailed shapes and fracture; both of these factors would be a useful addition to this system. A more complete surface model that maintained even sampling would be required for these methods to be applied. Likewise, further optimizations to allow the system to either provide faster updates or work with larger landscapes would be unquestionably useful. One method of improving the speed of updates might be to take direct advantage of specialized hardware (GPUs or physics processors); the current implementation performs all calculations on the main CPU.

My implementation only scratches the surface of plate tectonic interaction. I have limited myself to the relatively straightforward problem of two-plate collisions; a fuller simulation could provide for plate spreading, the consumption and extrusion of lithospheric rock by the athenosphere, transform faults, and systems of multiple plates. I have not made any effort to simulate volcanic activity, a consequence of plate subduction that has produced striking landscape features in the real world. Ideally, a system such as this would eventually be extended to simulate the plate interactions of an entire planet.

Simulating plate tectonics for computer graphics purposes is a new topic, and this project presents only a first attempt. It is likely that more specialized methods could provide better results than the very general object-deformation model I have used.

References

[1] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, M. Alexa: *Point based animation of elastic, plastic, and melting objects.* Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Computer Animation. Aug 2004.

[2] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, M. Gross: *Contact handling for deformable point-based objects.* Proceedings of Vision, Modeling, Visualization VMV'04. Nov 2004, pp 339-347.

[3] M. Pauly, R. Keiser, B. Adams, P. Dutre, M. Gross, L. Guibas: *Meshless animation of fracturing solids.* ACM Transactions on Graphics(TOG). Volume 24, Issue 3, July 2005, pp 957-964.

[4] J. F. O'Brien, A. W. Bargteil, J. K. Hodgins: *Graphical modeling and animation of ductile fracture.* Proceedings of SIGGRAPH 2002. October 2002, pp 291-294.

[5] M. Pauly, M. Gross, L. P. Kobbelt: *Efficient simplification of point-sampled surfaces.* Proceedings of the conference on Visualization '02. 2002, pp 163-170.

[6] M. Müller, M. Gross: *Interactive virtual materials.* Proceedings of the 2004 conference on Graphics interface GI '04. May 2004, pp 239-246.

[7] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranerts, M. Gross: *Optimized spatial hashing for collision detection of deformable objects*. In Proc. Vision, Modeling, Visualization VMV (2003), pp 47-54.

[8] P. Volino, M. Courchesne, N. M. Thalmann: *Versatile and efficient techniques for simulating cloth and other deformable objects*. Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. September 1995, pp 137-144.

[9] M. Desbrun, M.-P. Cani: *Animating soft substances with implicit surfaces*. Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. September 1995, pp 287-290.

[10] M. Desbrun, M.-P. Cani: *Smoothed particles: A new paradigm for animating highly deformable bodies.* 6th Eurographics Workshop on Computer Animation and Simulation '96. 1996, pp 61-76.

[11] A. Cox. *Plate Tectonics: How It Works*: Blackwell Scientific Publications, Inc. Boston: 1986.

[12] P. Lindeberg, M. Kiger, J. Russell: *This Dynamic Earth: The story of plate tectonics.* U.S. Geological Survey, Dec 2001. http://pubs.usgs.gov/publications/text/dynamic.html. Jan 27, 2006.

[13] F. K. Musgrave, C. E. Kolb, R. S. Mace: *The synthesis and rendering of eroded fractal terrains.* Proceedings of the 16th annual conference oncComputer graphics and interactive techniques, 1989. pp 41-50.

[14] A. Kelley, M. Malin, G. Nielson: *Terrain simulation using a model of stream erosion.* Proceedings of the 15th annual conference on Computer graphics and interactive techniques, 1988. pp 263-268.

[15] B. Benes; R. Forsbach: *Layered data representation for visual simulation of terrain erosion*, Spring Conference on Computer Graphics, 2001, pp 80-86.

[16] M. Pauly, R. Keiser, L. P Kobbelt, M. Gross: *Shape modeling with point-sampled geometry*, ACM Transactions on Graphics. Volume 22, Issue 3, July 2003, pp 641-650.

[17] D. Terzopoulos, K. Fleischer: *Modeling inelastic deformation: viscoelasticity, plasticity, fracture*, ACM SIGGRAPH Computer Graphics, Proceedings of the 15th annual conference on Computer graphics and interactive techniques. Volume 22, Issue 4, June 1988, pp269-278.

[18] Benoit B. Mandelbrot: *The Fractal Geometry of Nature*, W. H. Freeman and Co., New York, 1982

[19] G. Thomas Mase, George E. Mase: *Continuum Mechanics for Engineers*, CRC Press, New York, 1999.

[20] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery: *Numerical Recipes in C: The Art of Scientific Computing*, second ed. Cambridge University Press, 1992. http://www.library.cornell.edu/nr/cbookcpdf.html

[21] Persistance of Vision Raytracer Pty. Ltd.: *Persistance of Vision Raytracer(POV-ray)*. http://www.povray.org, 11/2/2006.

[22] ImageMagick Studio LLC: *ImageMagick* and *Magick++*. http://www.imagemagick.org, 5/17/2006.

[23] The GNU project: *GNU Scientific Library (gsl)*. http://www.gnu.org/software/gsl/, 5/17/2006.

[24] M. Garland: *libgfx Graphics Support Library*. http://graphics.cs.uiuc.edu/~garland/software/libgfx.html, 5/17/2006

A User Documentation

A.1 Building

This software has been compiled and run successfully on Solaris 9 and Gentoo Linux. It should work on any UNIX-based system with the required libraries installed. The tectonic simulator requires a number of external libraries, including:

- ImageMagick built with Magick++ support, v6.2.5
- GNU Scientific Library (aka libgsl), v1.8
- libgfx v1.1.0

All of these are freely available on the web; see the References above for the appropriate URLs. The versions listed are those with which the software has been tested; later versions will probably still work, while earlier ones may or may not do so.

Depending on where these dependencies are installed, you will need to edit the makefile to set the variables INCFLAGS and LDFLAGS to the correct directories. If all the requirements are installed in their default locations on your system, you should be able to simply comment these out. If not, you'll need to install local copies of those libraries and point these variables to the appropriate 'include' and 'lib' directories.

Once you have the libraries set up correctly, *make all* (or simply *make*) will compile the software. The executable itself will be named *tectonic*.

A.2 Running

The *tectonic* executable accepts a number of command line options that allow you to specify most of the parameters discussed above. These options can be specified in any order; later options will take precedence over earlier ones if they conflict.

- *-d* : Enable recalculation of density and volume after each time step.
- *-f <filename>* : Read initial landscape from image file at <filename>
- *-t <steps>* : Stop the simulation after <steps> updates.
- *-m <phyxel mass>* : Set phyxel mass for both objects.
- *-rm/-bm <phyxel mass>*: Set phyxel mass for only the (r)ed or (b)lue plate.
- *-ry/-by <modulus>*: Set Young's modulus for the (r)ed or (b)lue plate.
- *-rp, -bp <ratio>*: Set Poisson's ratio for the (r)ed or (b)lue plate. This ratio is between 0 and 0.5 for most materials; certain very strange substances have a negative Poisson's ratio, but none have one greater than 0.5.

- *-vx <relative plate force x>*: Set the relative collision speed along the x-axis (left-right).
- *-vy <relative plate force y>*: Set the relative collision speed along the y-axis (up-down).
- *-vz <relative plate force z>*: Set the relative collision speed along the z-axis (in-out).
- *-k <collision force constant>*: Multiply the default k_s by <collision force constant>.
- *-g <grid cell size>*: Set the size of the grid cells for spatial hashing.
- *-h <phyxel support radius>*: Set the phyxel support radius.
- *-s*: Enable surface representation. This option is required for meaningful height field output.
- *-o <output filename>* : Change the output filename; default "heights.png"
- *-b <interval>* : Run in batch mode: don't create an OpenGL view, and output a height field every <interval> steps. The names of files written in this mode have the current number of time steps prepended to keep them separate. The *-b* option will assume a time limit of 200 steps if you don't specify otherwise with *-t*.

A.3 Controlling

When running and not in batch mode, *tectonic* allows you to control its behavior in a limited fashion with a few key presses. Recognized keystrokes are:

- ' ' (*space bar*): Pause or unpause the simulation.
- '/': Toggle phyxel coloring mode between object-based and cell-based.
- 's': Pauses the simulation, and saves a height field snapshot.
- '?': Prints a bunch of debugging information.
- 'v': Toggles verbose output of collision forces (more debugging information).

B Further Examples

I include here a number of additional image sequences (figures 25 onward), created using various combinations of the parameters and input files described above. As with those results discussed above, these were run with default parameters except where otherwise specified.

Figure 26: Island landscape collision with a massive subducting plate (rm=12000, bm=8000)

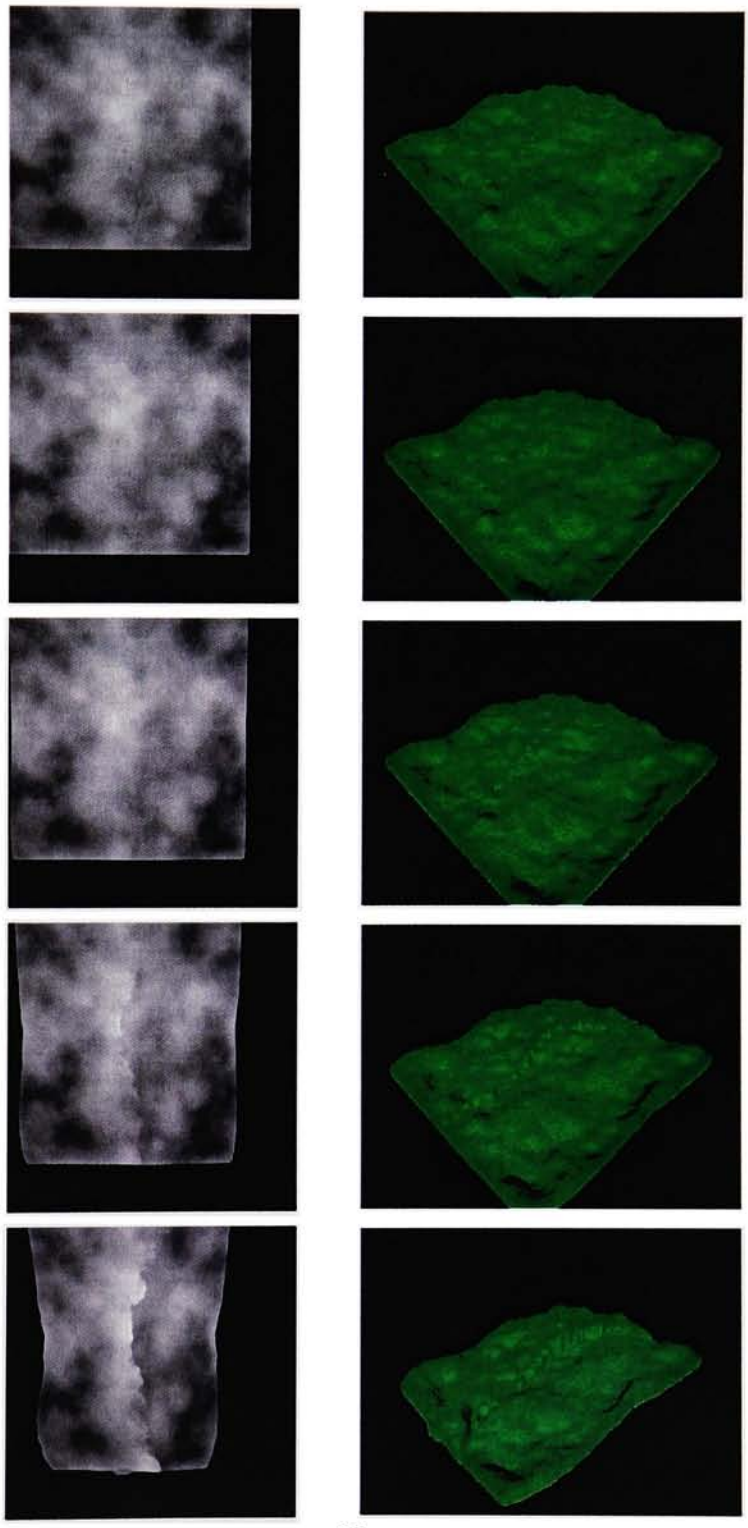


Figure 28: Island landscape collision with a light subducting plate (rm=8000, bm=12000)

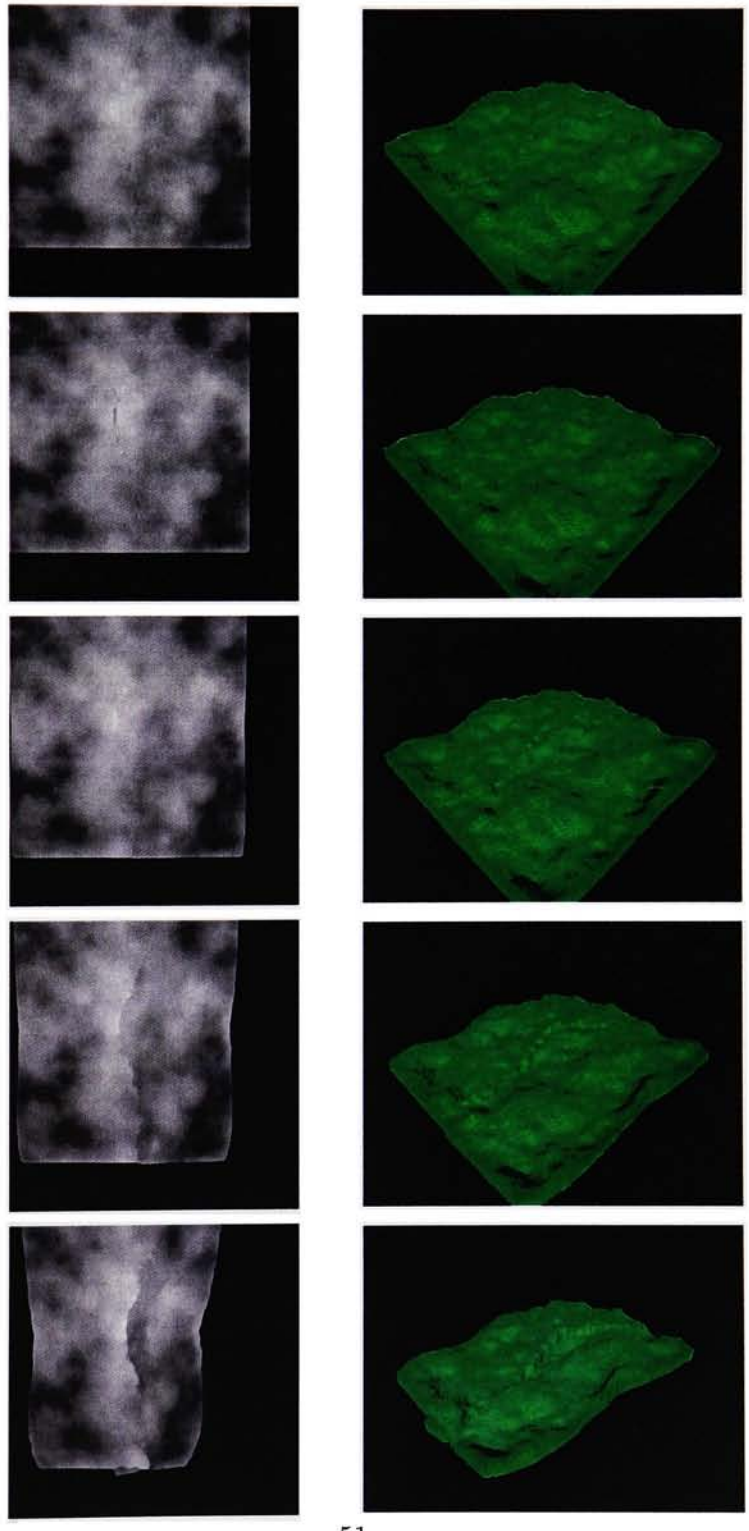


Figure 27: Shore landscape collision with massive subducting plate.

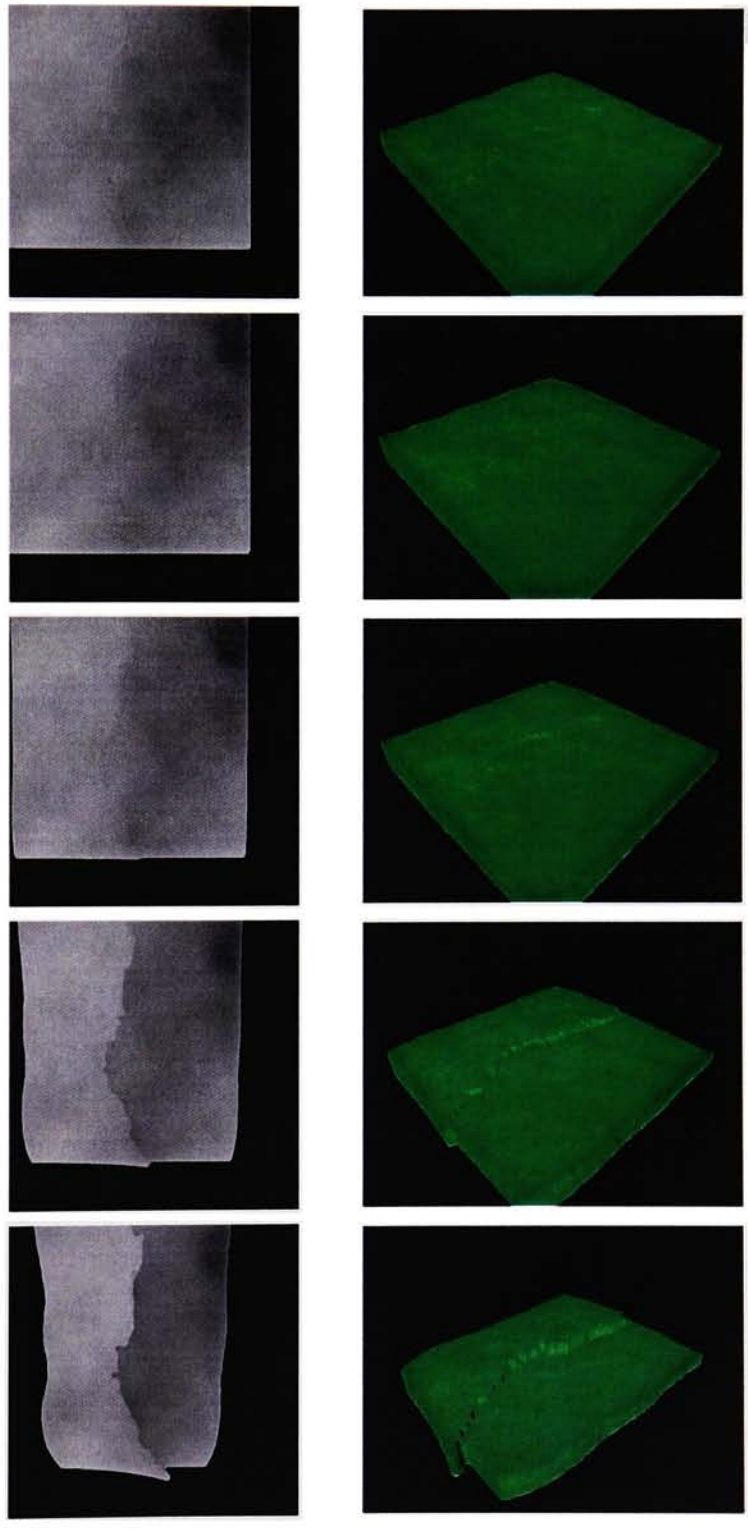


Figure 30: Island landscape, slow collision (k_s and $\mathbf{g}$ each $1/2$ normal)

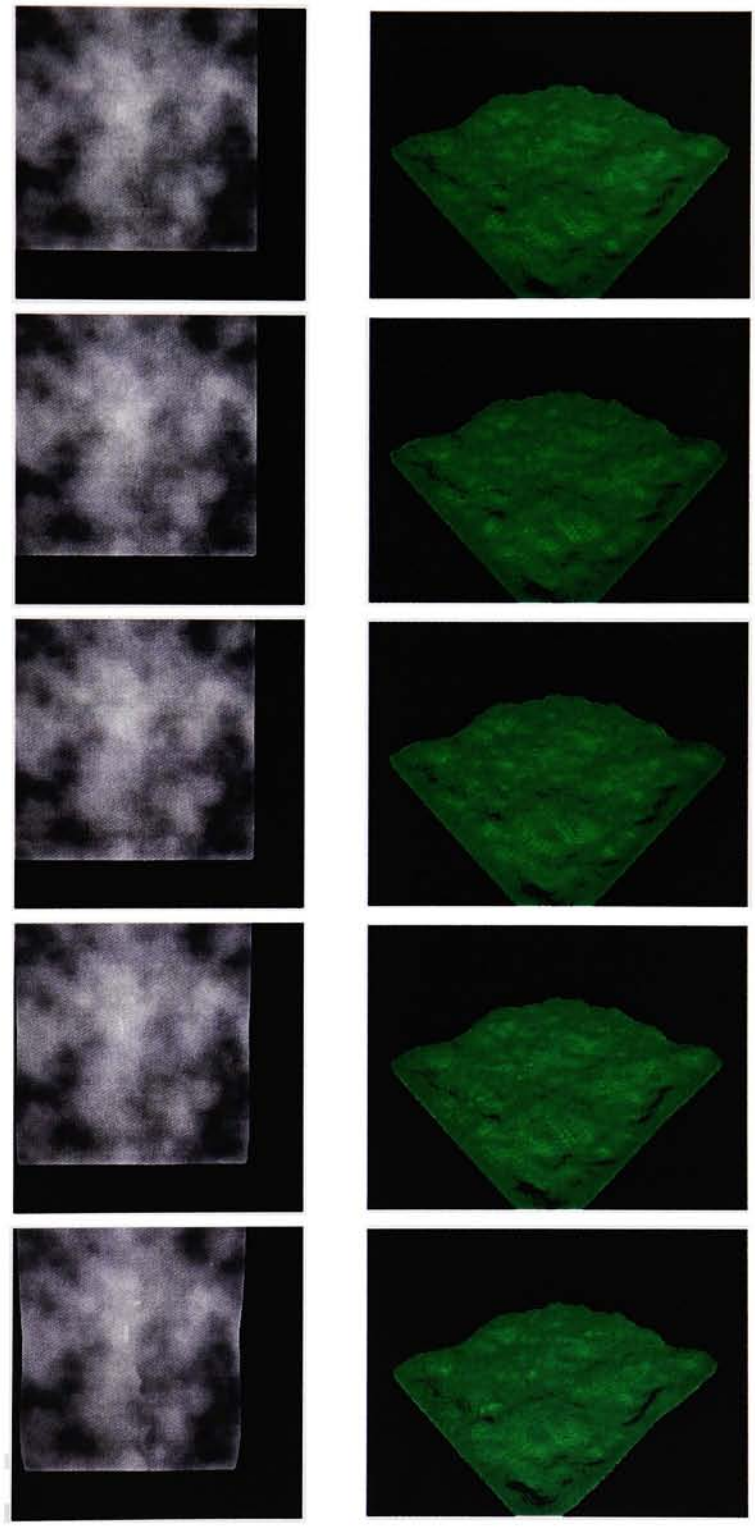


Figure 29: Shore landscape collision with light subducting plate.

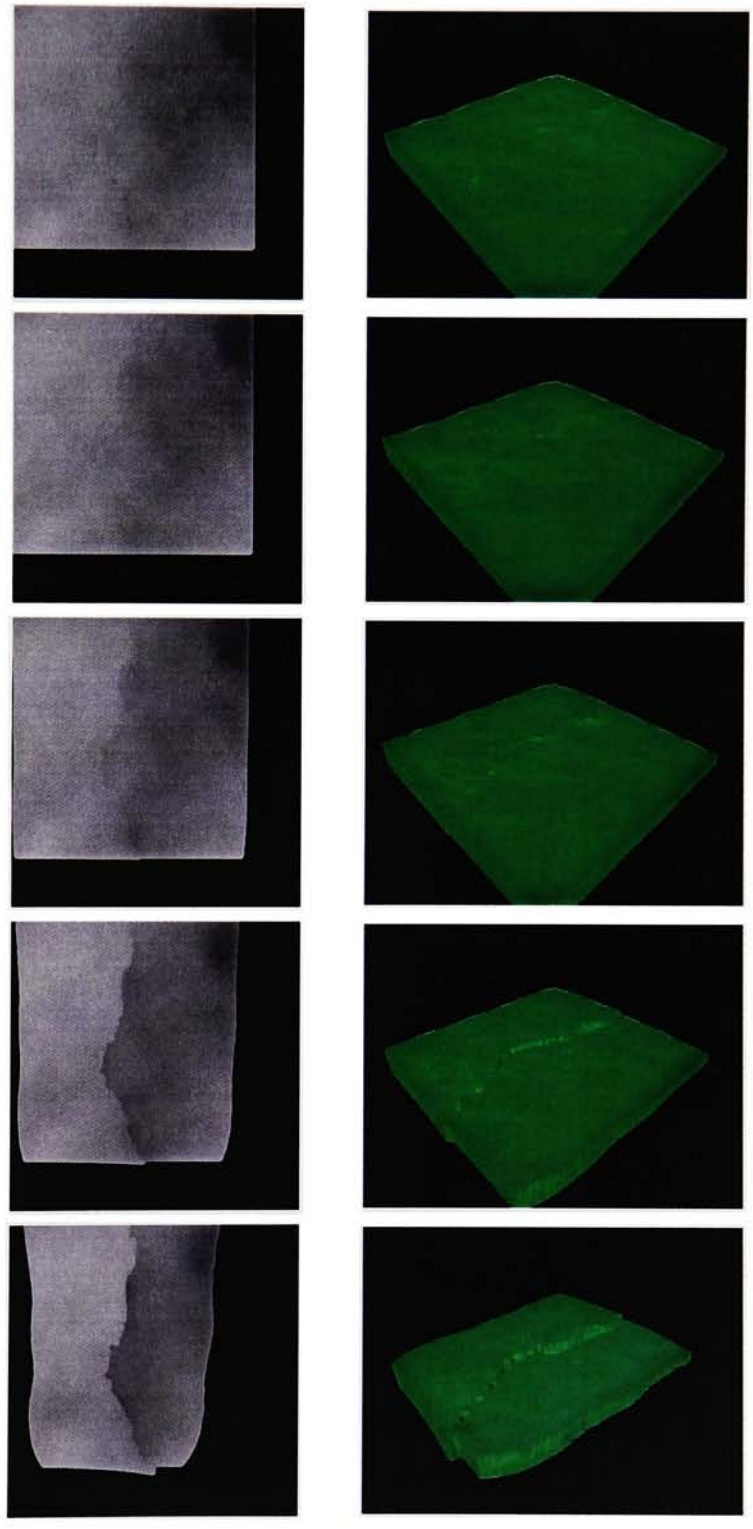


Figure 32: Shore landscape, angled collision with $h = 3$

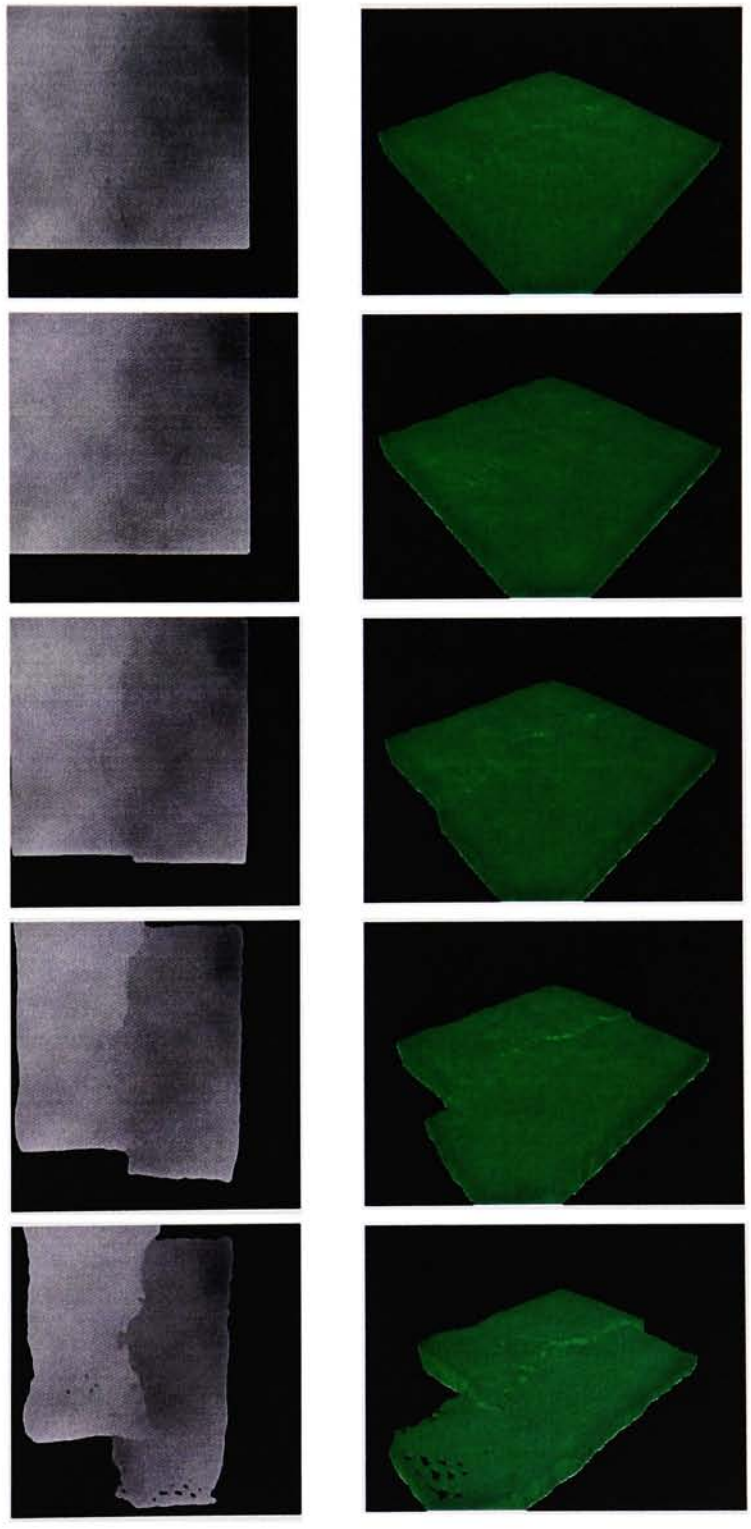


Figure 31: Mountain landscape, slow collision.

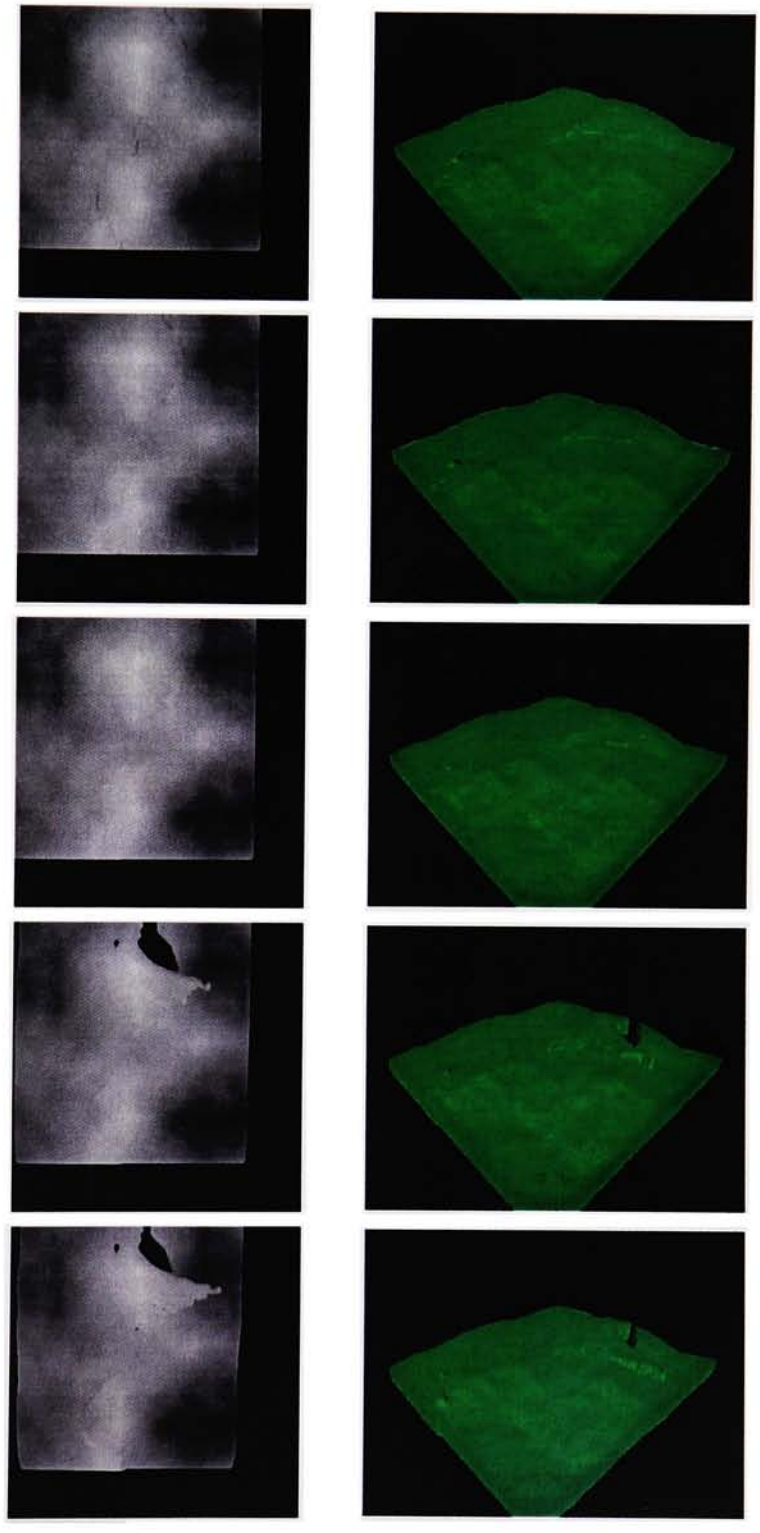


Figure 33: Shore landscape, opposite angled collision with $h = 3$