An Interactive Simulation Environment for Craniofacial Surgical Procedures

Dan Morris[1] Sabine Girod[2] Federico Barbagli[1] Kenneth Salisbury[1]
[1]Department of Computer Science and [2]Division of Plastic and Reconstructive Surgery, Stanford University
{dmorris, barbagli, jks}@robotics.stanford.edu, sgirod@stanford.edu

Abstract. Recent advances in medical imaging and surgical techniques have made possible the correction of severe facial deformities and fractures. Surgical correction techniques often involve the direct manipulation – both relocation and surgical fracture – of the underlying facial bone. The work presented here introduces an environment for interactive, visuohaptic simulation of craniofacial surgical procedures, with an emphasis on both mandibular distraction procedures and traditional orthognathic surgeries. The simulator is intended both for instruction and for procedure-specific rehearsal, and can thus load canonical training cases or patient-specific image data into the interactive environment. A network module allows remote demonstration of procedure technique, a form of 'haptic tutoring'.

This paper discusses the simulation, haptic feedback, and graphic rendering techniques used to drive the environment. Particular emphasis is placed on techniques for fracture and subsequent rigid manipulation of bone structures, a key component of the relevant procedures.

1. Introduction

1.1 Surgical Background

Incorrect alignment of the jaws – due to congenital malformation, trauma, or disease – can result in cosmetic deformation and problems with chewing and/or breathing. Orthognathic surgeries correct such problems, typically by inducing a fracture in one or both jaws (generally using a bone saw), displacing the fractured components into an anatomically preferable configuration, and installing bone screws and/or metal plates to fix the bone segments in their new positions.

This approach is often prohibited by the severity of the deformation, the size of the separation that would be required after fracture, or the sensitivity of the surrounding soft tissue. In these cases, distraction osteogenesis is often employed as an alternative. Here a similar procedure is performed, by only a minor separation is created intraoperatively. Instead of spanning the gap with a rigid plate, an adjustable distractor is fixed to the bone on both sides of the gap. The distractor can be used to gradually widen the fracture over a period of several weeks, allowing accommodation in the surrounding tissue and allowing the bone to heal naturally across the fracture.

These procedures are likely to benefit from surgical simulation for several reasons. The complex, patient-specific planning process and the significant anatomic variation from case to case suggests that an end-to-end simulator will assist physicians in preparing for specific cases. Furthermore, distraction procedures have been introduced to the craniofacial surgical community only within the last ten to fifteen years, and an effective simulator will significantly aid in the training and re-training of this new class of procedures, and with the exploration of alternative techniques for effective surgeries.

1.2 Previous work

Several projects [1,2,3] have focused on the simulation of craniofacial surgical procedures, although the focus of most previous work has been on the prediction of soft tissue movement and the prediction of post-operative facial appearance given a series of bone manipulations. The surgical procedure itself is typically represented as a series of geometric operations – defined via explicit cutting planes or analytically-specified transformations.

Other work has focused on simulation of dental [4] and otologic [5,6,7,8] procedures, which are similar to the surgeries discussed here in terms of haptic feedback and bone drilling/cutting. However, the procedures presented here additionally require the rigid manipulation of bone fragments and the attachment of rigid structures (plates, etc.) to the bone.

2. Methods

2.1 Data sources and preprocessing

FIGURE 1 summarizes the preprocessing stages that transform image data into the format used for interactive rendering. Isosurfaces are first generated from CT or MR data using the marching cubes method [9]; this allows us to discard regions of data that are not part of the skull. This algorithm does not generate closed meshes for all inputs, so we cap any holes in the resulting isosurfaces using the 3d Studio Max software package (Discreet Inc., Montreal, Quebec). A non-repeating set of texture coordinates is then generated on the isosurface mesh, to be used later for interactive rendering.

A flood-filling technique is then used to build a voxel grid from the isosurface data, using an AABB tree to accelerate the numerous collision tests required at this stage. For voxels situated at the boundary of the bone volume, we find the nearest triangle to the voxel center and use barycentric interpolation to assign texture coordinates and surface normals to those voxels. The resulting voxel array is exported along with density information for all voxels. This voxel array is used directly for haptic rendering, and is also retessellated into a new surface used for interactive graphic rendering (see SECTION 2.2). Overall preprocessing time for a typical head CT data set is on the order of fifteen to twenty minutes.

Models of bone plates have been supplied by the manufacturer, and are loaded into our environment directly as surface meshes.

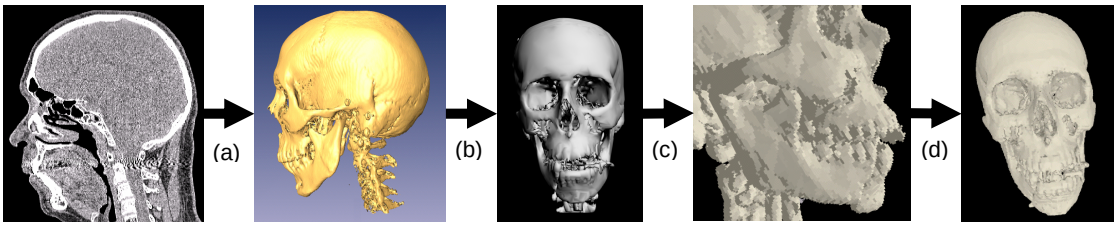


FIGURE 1: A summary of our preprocessing pipeline. CT data sets are (a) isosurfaced, (b) capped and smoothed, (c) flood-filled to generate a voxel array, and (d) re-tessellated into a final surface mesh.

2.2 Interactive Rendering

In order to leverage previous work in haptic rendering of volumetric data [10] while still maintaining the benefits of surface rendering in terms of hardware acceleration and visual effects, we maintain a hybrid data structure in which volumetric data are used for haptic rendering and traditional triangle arrays are used for graphic rendering. In order to simplify and accelerate the process of updating our polygonal data when the bone is modified, we build a new surface mesh – in which vertices correspond directly to bone voxels – rather than using the original isosurface mesh.

The voxel array representing the bone model is loaded into our simulation environment, and a polygonal surface mesh is generated to enclose the voxel grid. This is accomplished by exhaustively triangulating the voxels on the surface of the bone region, i.e.:

```
for each voxel v1 that is on the bone surface
  for each of v1's neighbors v2 that is on the bone surface
    for each of v2's neighbors v3 that is on the bone surface
      generate a triangle (v1,v2,v3) oriented away from the bone surface
```

Here being 'on the bone surface' is defined as having non-zero bone density and having at least one neighbor that has no bone density. Although this generates a significant number of triangles (on the order of 200,000 for a typical CT data set), we use several techniques to minimize the number of triangles that are generated and/or rendered. To avoid generating duplicate triangles, each voxel is assigned an index before tessellation, and triangles are rejected if they do not appear in sorted order. A second pass over the mesh uses the observations presented in [11] to eliminate subsurface triangles that will not be visible from outside the mesh.

To take advantage of the fact that the user does not frequently change the simulation's perspective, we maintain two triangle arrays, one containing the complete tessellation of the current bone volume (the "complete array"), and one containing only those that are visible from positions close to the current camera position (the "visible array"). The latter array is initialized at startup and any time the camera comes to rest after a period of movement. Visible triangles are those with at least one vertex whose normal points towards (less than 90 degrees away from) the camera. Because this visibility-testing pass is time-consuming, it is performed in the background; the complete array is used for rendering the scene during periods of camera movement (when the visible array is considered 'dirty') and during the reinitialization of the 'visible' array.

As a final optimization, we use the nvtristrip library [12] to reorder our triangle and vertex arrays for optimal rendering performance. We could have further reduced rendering time by generating triangle strips from our triangle lists, but this would add significant complexity to the process of updating the surface mesh to reflect changes to the underlying voxel grid.

2.3 Interactive Tools

In our environment, the user makes use of one or two SensAble Phantom [13] haptic devices – providing six-degree-of-freedom control of each tool and three-degree-of-freedom haptic feedback – to control several tools that can interact with the bone data.

The primary bone modification tools are the drill/saw tools, which use the method presented in [10] to perform haptic rendering and bone removal for tools of varying size and shape. FIGURE 2 displays a drill tool being used to remove bone density. See [10] for a complete description of their modified Voxmap-Pointshell algorithm; only a summary will be provided here. In short, a series of sample points is distributed on the surface of the tool; at each haptic iteration, each sample is tested for immersion in the bone volume. A ray is traced from each immersed sample point toward the tool center until it reaches either the tool center or the bone surface. Bone density is removed for each voxel these rays pass through, and a haptic force is generated along each ray to push immersed points out of the bone volume.

When bone voxels are removed from our environment, our hybrid data structure requires that the area around the removed bone be retessellated. Consequently, bone voxels are queued up by our haptic rendering thread as they are removed, and the graphic rendering thread retessellates the region around each voxel pulled from this queue. That is, for each removed voxel, we see which of his neighbors have been "revealed" and create triangles that contain the centers of these new voxels as vertices. Specifically, for each removed voxel v, we perform the following steps:

```
for each voxel v′ that is adjacent to v and on the bone surface
  if a vertex has not already been created to represented v′
    create a vertex representing v′ and compute the surface gradient at v′
    queue v′ for triangle creation
for each voxel v′ that is "queued for triangle creation"
  generate triangles adjacent to v′ (see below)
```

Once again, a voxel that is "on the bone surface" has a non-zero bone density and has at least one neighboring voxel that contains no bone density. When all local voxels have been tested for visibility (i.e. when the first loop is complete in the above pseudocode), all new vertices are fed to a

triangle generation routine. This routine finds new triangles that can be constructed from new vertices and their neighbors, orients those triangles to match the vertices' surface normals, and copies *visible* triangles to the "visible triangle array" (see SECTION 2.2). The reason for "queuing triangles for triangle creation" is that the generation of triangles – performed in the second loop above – depends on knowing which local voxels are visible, which is only known after the completion of the first loop.

An additional bone modification tool allows the introduction of large bone cuts via a planar cut tool (see FIGURE 4). This tool generates no haptic feedback and is not intended to replicate a physical tool. Rather, it addresses the need of advanced users to make rapid cuts for demonstration or for the creation of training scenarios. Bone removal with this tool is implemented by discretizing the planar area – controlled in six degrees of freedom – into voxel-sized sample areas, and tracing a ray a small distance from each sample along the normal to the plane. This ray is identical to the rays traced from the surface of a drill-type tool, discussed in more detail above. No haptic feedback is generated, and each ray is given infinite "drilling power", i.e. all density is removed from any voxels through which each ray passes. The distance traced along each ray is controlled by the user. This allows the user to remove a planar or box-shaped region of bone density, demonstrated in FIGURE 4.

A final set of tools allows the user to manipulate models of several distractors and/or Synthes bone plates. The inclusion of these plate models allows users to plan and practice plate-insertion operations interactively, using industry-standard plates. Collision detection for haptic feedback is also performed using a set of sample points; in this case, the sample points are generated by sampling 100 vertices of each model and extruding them slightly along their normals (because these models tend to be very thin relative to our voxel dimensions) (FIGURE 3). Since there is no well-defined tool center toward which we can trace rays for penetration calculation, rays are traced along the model's surface normal at each sample point. At any time, the user can rigidly affix a plate tool to a bone object with which it is in contact. Future work will include a simulation of the bone-screw-insertion process.

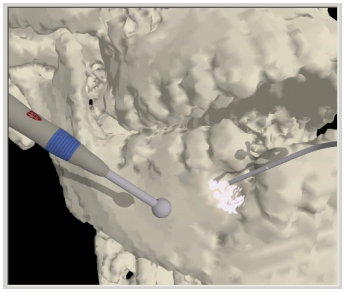


FIGURE 2: Using the drill tool to interactively modify bone volume.

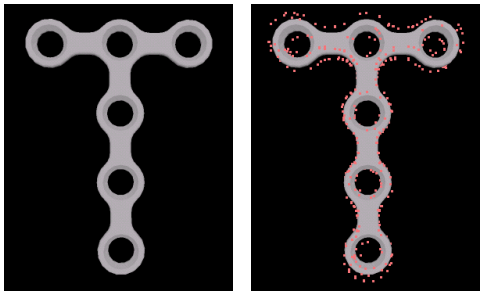
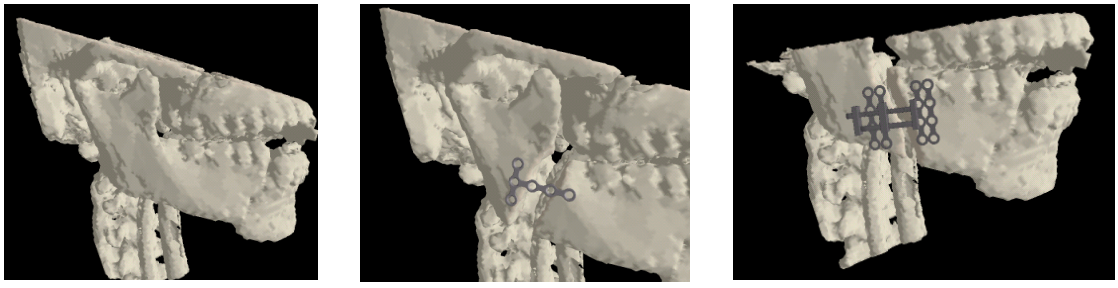


FIGURE 3: Sampling and extrusion of the surface of a bone plate for collision detection with bone.



(a) (b) (c)

FIGURE 4: Binding of bone plates to a bone model. (a) Bone model before fracture (a planar cut has been applied to simplify the image for demonstration). (b) A standard bone plate fixed across a fracture. (c) A distractor fixed on both sides of a fracture.

2.4 Continuity detection

A critical step in simulating craniofacial procedures is the detection of cuts in the bone volume that separate one region of bone from another, thus allowing independent rigid transformations to be applied to the isolated bone segments.

In our environment, a background thread performs a repeated flood-filling operation on each bone structure. A random voxel is selected as a seed point for each bone object, and flood-filling proceeds through all voxel neighbors that currently contain bone density. Each voxel maintains a flag indicating whether or not it has been reached by the flood-filling operation; at the end of a filling pass, all unmarked voxels (which must have become separated from the seed point) are collected and moved into a new bone object, along with their corresponding data in the vertex and triangle arrays.

FIGURE 5 displays a bone object that has been cut and the subsequent independent movement of the two resulting structures. Here – for demonstration – the cut-plane tool is used to create the fracture; during simulated procedures, most fractures will likely be created by the drilling/sawing tools.

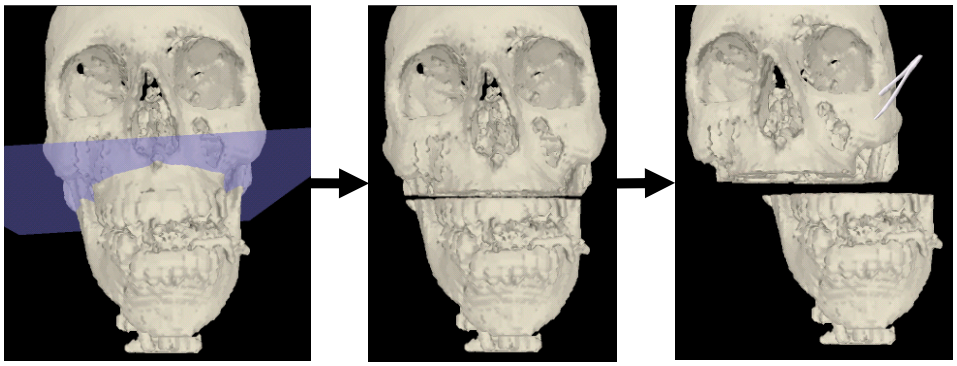


FIGURE 5: The use of the cut-plane tool and the independent manipulation of discontinuous bone regions. (a) The cut-plane tool is used to geometrically specify a set of voxels to remove. (b) The volume after voxel removal. (c) The flood-filling thread has recognized the discontinuity, and the bone segments can now be manipulated independently.

2.5 Network Training

Surgical training is typically focused on visual observation of experienced surgeons and verbal descriptions of proper technique; it is impossible for a surgeon to physically demonstrate the correct 'feel' of bone manipulation with physical tools. With that in mind, we have incorporated a 'haptic tutoring' module into our environment, allowing a trainee to experience forces that are the result of a *remote* user's interaction with the bone model.

Ideally, the trainee would experience both the movements of the instructor's tool and the force applied to/by the instructor, but it is difficult to control both the position and the force at a haptic end-effector without any control of the compliance of the user's hand. To address this issue, we bind the position of the trainee's tool to that of the instructor's tool via a low-gain spring, and add the resulting forces to a 'playback' of the forces generated at the instructor's tool. I.e.:

$$F_{trainee} = K_p(P_{trainee} - P_{instructor}) + F_{instructor}$$

…where $F_{instructor}$ and $F_{trainee}$ are the forces applied to the instructor's and trainee's tools, and $P_{instructor}$ and $P_{trainee}$ are the position of the instructor's and trainee's tools. K_p is small enough that it does not interfere significantly with the perception of the high-frequency components transferred from the instructor's tool to the trainee's tool, but large enough that the trainee's tool stays in the vicinity of the instructor's tool. In practice, the error in this low-gain position controller is still within reasonable visual bounds, and the trainee perceives that he is experiencing the same force *and* position trajectory as the instructor.

2.5 Neurophysiology simulation

Another goal of our simulation environment is to train the surgical skills required to avoid critical and/or sensitive structures when using potentially dangerous tools. The inferior alveolar nerve is at particular risk during most of the procedures this environment is targeting. We thus incorporate a virtual nerve monitor that presents the user with a representation of the activity of nerve bundles in the vicinity of the procedure (FIGURE 6). Nerves are currently placed explicitly for training scenarios; future work will include automatic segmentation of large nerves from image data.

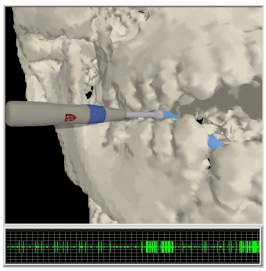


FIGURE 6: Interactive monitoring of virtual nerves. When the virtual tool makes contact with the bone in the vicinity of the virtual nerve bundle (highlighted here in blue), a series of neural pulses is presented via graphic and auditory monitoring tools. Several such bursts are displayed in this clip, captured from the graphic monitor.

3. Discussion and Conclusion

Initial work with surgeons indicates that the haptic feedback associated with bone manipulation is quite realistic. Future work on this environment will thus focus on the incorporation of soft tissue data into the simulation. Online soft tissue representations will be critical for the simulation of constraints imposed on bone movements, and for the training of complete procedures including incisions and soft tissue displacement. Offline soft tissue simulation will allow us to incorporate previous work on the prediction of soft-tissue appearance following craniofacial surgery (e.g. [1],[2],[3]). Furthermore, we hope to extend our training of sensitive structure avoidance; a representation of potentially sensitive blood vessels will be valuable, as will the automated or semi-automated extraction of such structures directly from image data.

Acknowledgements

We thank Doug Wilson for his implementation of shadowing and bone dust, Nik Blevins for his drill models, and Synthes for providing plate models. Support for this work was provided by NIH grant LM07295, the AO Foundation, and the NDSEG fellowship.

References

[1] Keeve E, Girod S, Kikinis R, Girod B. Deformable modeling of facial tissue for craniofacial surgery simulation. Computer Aided Surgery, 1998, 3:228–38.
[2] Schmidt JG, Berti G, Fingberg J, Cao J, Wollny G. A Finite Element Based Tool Chain for the Planning and Simulation of Maxillo-Facial Surgery. Proceedings of the fourth ECCOMAS, Jyvaskyla, Finland, 2004.
[3] Koch RM, Roth SHM, Gross MH, A. Zimmermann P, Sailer HF. A Framework for Facial Surgery Simulation. Proc of the 18th Spring Conference on Computer Graphics, p33–42, 2002.
[4] Thomas G, Johnson L, Dow S, Stanford C. The design and testing of a force feedback dental simulator. Computer Methods and Programs in Biomedicine. 2001 Jan;64(1):53-64.
[5] Morris D, Sewell C, Blevins N, Barbagli F, Salisbury K. A Collaborative Virtual Environment for the Simulation of Temporal Bone Surgery. Proceedings of MICCAI 2004, v. II pp. 319-327.
[6] Agus M, Giachetti A, Gobbetti E, Zanetti G, John NW, Stone RJ: Mastoidectomy simulation with combined visual and haptic feedback. Proceedings of MMVR 2002.
[7] Bryan J, Stredney D, Wiet G, Sessanna D. Virtual Temporal Bone Dissection: A Case Study. Proceedings of IEEE Visualization 2001, Ertl et. Al., (Eds): 497-500, October 2001.
[8] Pflesser B, Petersik A, Tiede U, Hohne KH, Leuwer R Volume cutting for virtual petrous bone surgery. Computer Aided Surgery 2002;7(2):74-83.
[9] Lorensen WE, Cline HE. Marching Cubes: A high resolution 3D surface construction algorithm. ACM Computer Graphics 1987, 21:163–169.
[10] Renz M, Preusche C, Potke M, Kriegel HP, Hirzinger G. Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. Proc Eurohaptics, p149-154, 2001.
[11] Bouvier, D.J. Double-Time Cubes: A Fast 3D Surface Construction Algorithm for Volume Visualization. Int'l Conf on Imaging Science, Systems, and Technology, June 1997.
[12] NVIDIA Corporation. nvtristrip library. February 2004. http://developer.nvidia.com/.
[13] Massie TH, Salisbury JK. The PHANTOM Haptic Interface: A Device for Probing Virtual Objects. Symposium on Haptic Interfaces for Virtual Environments, Chicago, IL, Nov 1994.