TECHNICAL REPORT: CONVEX HULL GAME: A TANGIBLE CONTEXT FOR ALGORITHMS AND COMPUTER GRAPHICS CONCEPTS

Zane R. Cochran
Berry College
Mount Berry, GA 30149
zane@zanecochran.com

Nadeem Abdul Hamid
Berry College
Mount Berry, GA 30149
nadeem@acm.org

ABSTRACT

This paper presents a unique way to engage students in an Algorithms or Computer Graphics course through the development of a convex hull-based game. The game provides a compelling context to use image processing techniques to create a simple tangible user interface that allows players to physically manipulate the input to a convex hull algorithm and display an impressive visualization as the result.

INTRODUCTION

Finding the convex hull of a set of points in the plane is a fundamental problem in computational geometry with applications in a variety of areas [9]. In a curricular context, Computer Science students may be exposed to the problem in an Algorithms or Computer Graphics course. (The ACM CS curriculum lists convex hull as an elective topic under both categories [1].**)** For the purposes of a computer graphics course, finding a convex hull might be presented as a basic problem to be solved to enable implementation of other image processing and graphics techniques [4]. Algorithms textbooks (e.g., [3, 7]) present convex hull as an example of a problem for which a recursive, divide-and-conquer strategy produces an efficient solution. In either case, the convex hull problem appears somewhat as a means to an end - an example demonstrating algorithm design technique and/or an intermediate problem to be solved and used in a larger context - not a problem with direct or immediate relevance to a user.

This paper presents a simple game based directly on the notion of a convex hull. With a webcam and programming language that provides a suitable library for reading and processing arrays of image pixels, a program can be developed that tracks and displays players' moves on a computer screen. Using a projector and some additional hardware at reasonable cost, the game interface can be enhanced to provide a much more immersive experience. In this application, computing the convex hull is the end goal of the software, not an incidental example or intermediate step. The output of the algorithm is visible in a real and tangible way. This realization of the problem provides an interesting and potentially motivating context for implementing a convex hull algorithm in Algorithms and Computer Graphics courses. The implementation described in this paper was in fact developed by the first author to fulfill the requirements of a programming project in an undergraduate Algorithms course.

Convex Hull: The Problem and the Game

Abstractly, the *convex hull* of a set *S* of points is the smallest convex set containing *S*. (A set of points in the plane is called *convex* if, for any two points in the set, the line segment between them lies entirely within the set.) An intuitive visualization of the concept is to think of the points as nails driven into a board. The convex hull is the shape formed by a rubber band that is stretched just enough to surround all the nails. The points on the convex hull (called *exterior points*) are those to which the rubber band adheres [7].

The game of convex hull, invented for this work, is played by two players, each of whom has a pile of colored game pieces (e.g., checkers pieces). Players take turns tossing one of their pieces onto a large, flat surface bounded by barriers along the edges. The object of the game is to always have all of one's pieces form part of the convex hull. A player loses the match if they toss their piece and it lands within the hull formed by the pieces already on the board, or if the opponent (or the player) tosses a piece which causes one or more of the player's pieces to become interior points of the hull. Intricacies of game play are enumerated later in the paper.

Related Work

As discussed in the introduction, the convex hull problem is a classical and well-studied problem in computational graphics. The work described in this paper provides an immediate *tangible user interface* (TUI) for a program that solves the convex hull problem. The idea of a TUI [6] is to give "physical and graspable" external representations to digital information, in contrast to a graphical user interface (GUI), by which users interact "remotely" with digital information using generic input/output devices. TUIs have been developed and investigated in a variety of contexts, including games [10]. One of the challenges of TUI development is design and implementation of a mapping from physical objects and their manipulation to digital computation and feedback in a meaningful and effective way [6]. In general, depending on the application for which a TUI is being designed, this can be non-trivial. The context presented in this paper is simple enough that students who have mastered programming concepts of a typical CS1/CS2 sequence should be able to implement it, with very compelling results.

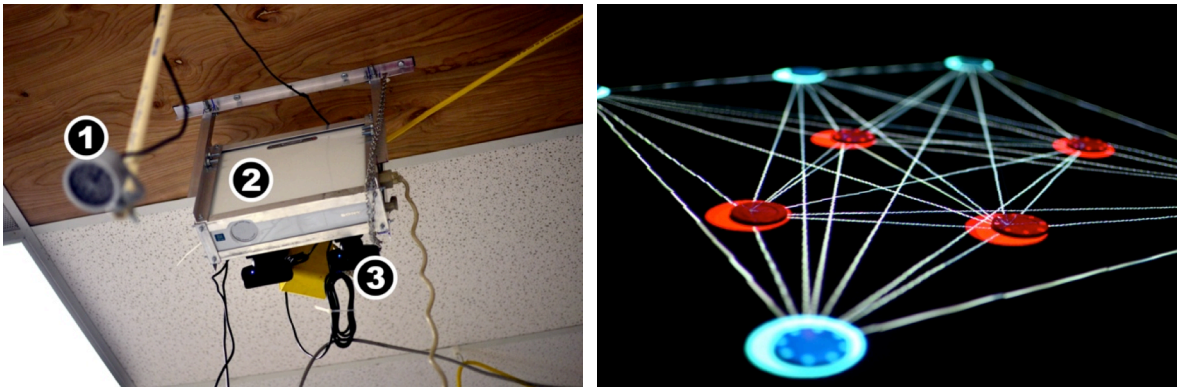
For the implementation described in what follows, Processing [9] was used. Processing is an environment created to teach fundamentals of computer programming in a media arts context. The underlying language of the environment is Java, so students who know Java or C will find it easy to start writing programs using it. While the authors have not investigated it, the tools developed to support the Media Computation [5] approach to introductory Computer Science may also provide a suitable environment for realizing a project such as the one described here.

CONVEX HULL GAME IMPLEMENTATION

The convex hull game provides a context in which students can integrate a variety of concepts to create a program that provides a tangible user interface (TUI) and turns a simple algorithm into a competitive game. The methods used for image processing, object detection, geometric transformations, etc. may be brute-force or more sophisticated, as appropriate. In this section, we describe hardware configurations options and outline the elements that need to be implemented in the corresponding software program.

Configuration	Advantages	Disadvantages
Simple	Low cost	Limited application
	Easy setup	Visualization limited to computer monitor
Augmented	Interactive visualization	Camera and projector required
	Varied application	Calibration necessary

Figure 1: Comparison of hardware configurations.



a. b.

Figure 2. (a) A typical Augmented Configuration hardware setup, including (1) infrared light source, (2) an LCD projector, and (3) an infrared camera, all attached to the ceiling above the playing area. (b) Picture of the playing surface in the Augmented Configuration; interior and exterior points and edges are projected directly onto the surface, creating an interactive augmented visualization.

Hardware Configuration

The hardware required for this exercise varies, depending on desired sophistication of the final product and level of proficiency with building support structures. Here we describe two configurations; the pros and cons of each are summarized in Figure 1. The Simple Configuration requires little hardware and focuses on input processing and convex hull computation, with simpler output rendering. The Augmented Configuration requires more hardware (which is inexpensive or should be available at a university) but the result is more impressive visually.

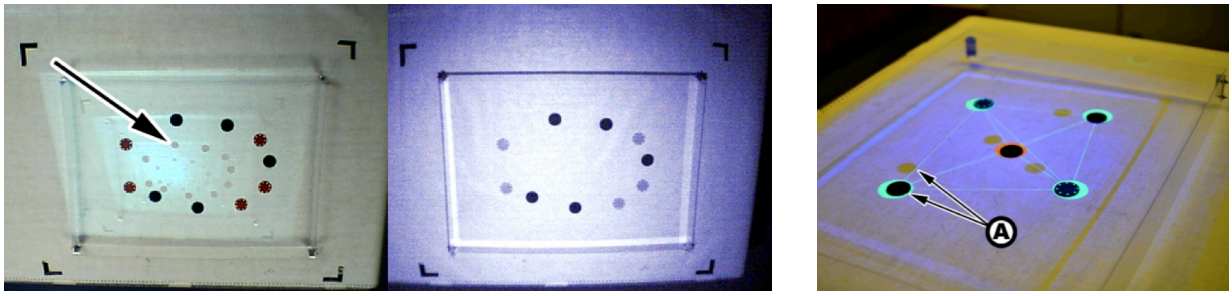
Simple Configuration

The Simple Configuration for the convex hull game consists of a video camera connected to a computer. The camera is placed so that it has a clear view of a flat playing surface. This setup is both inexpensive and portable and allows students to implement the game away from a laboratory environment. The primary drawback of this setup is that the output visualization is limited to the monitor of the computer running the program. Nonetheless, it provides a starting point to familiarize oneself with the necessary aspects of image processing and object detection.

Augmented Configuration

The Augmented Configuration of the game requires more effort setting up the hardware, but once configured, it could prove to be useful for a variety of additional projects. The basic requirements for this configuration consist of an infrared camera, an infrared light source, and a projector. For our setup, a PlayStation Eye camera modified with infrared filter and an infrared spotlight worked well. Ideally, the projector is mounted to the ceiling and angled downward to project onto a flat table below (Figure 2a). The camera and infrared light source are mounted so their field of vision corresponds as closely as possible to the projected image.

With this configuration, it is possible to not only detect objects on the playing surface, but to project an overlay of the current placement of pieces and the edges of the convex hull directly onto the surface, allowing players to enjoy immediate visual feedback of their game (Figure 2b). The infrared light source and camera is necessary for object detection to be effective; otherwise a feedback loop occurs in which the images emitted from the projector interfere with detection of the actual physical playing (Figure 3a, left). Because the projector emits visible light, the modified camera does not “see” the projected image (Figure 3a, right).



a. b.

Figure 3. (a) Comparison of projected light feedback as seen by a color camera (left) and infrared camera (right). Visible light (including projected image with dots) is almost completely filtered by the infrared camera. (b) A calibrated TUI with projected infrared video from camera using the Augmented Configuration. The output here displays the discrepancy (A) between the camera's field of view and the image emitted by the projector. Without calibration, the blue and red markers and edge lines would not overlap the physical game pieces.

Software Components

For the implementation of this work, Processing was chosen as the language and environment. While any language or library that facilitates capturing images from a video camera and manipulating pixel arrays can be used, Processing comes with libraries that allow students to retrieve still images from an attached camera and manipulate graphics objects with relative ease, enabling development of the program's three main components: object detection, finding the convex hull, and rendering output. In the subsections that follow, we outline aspects of each component, for both the simple and augmented hardware configurations.

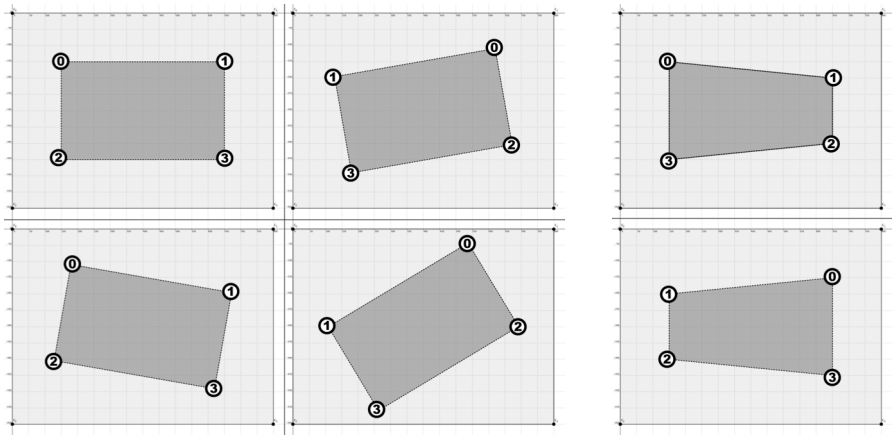
Object Detection

For the Simple Configuration, a reference image of the blank playing surface is captured as a two-dimensional array of pixels. Each subsequent frame of video is similarly stored as an array of pixels. The object detection algorithm compares each pixel of the reference array to the current frame array looking for significant differences in color values. (This is a simple approach to *background subtraction*.) Because each physical object appears as a blob of pixels in the image, a method is needed to determine when a detected pixel is part of a new object or a previously detected object. A straightforward way is to assume that every object (playing piece) is of a fixed size, so all pixels within a radius of that size belong to one object.

With respect to object detection, the Augmented Configuration is no different. This is the case because the infrared image in the Augmented Configuration only captures images of physical objects and not the projected image on the surface. Depending on the students' abilities, the image processing functions are provided as a library or written by the students themselves.

Identifying a Convex Hull

Once a collection of detected objects has been created, the next task is to identify the convex hull (and interior points) created by those objects. For the first three objects this is trivial as there can be no interior points. For more objects, however, it is necessary to distinguish exterior and interior points and label them for use in the rendering of the output. For the purposes of this game, we discovered that any convex hull algorithm will work, even a brute-force approach, and developing this algorithm can be part of the student exercise. Runtime efficiency



a. b.

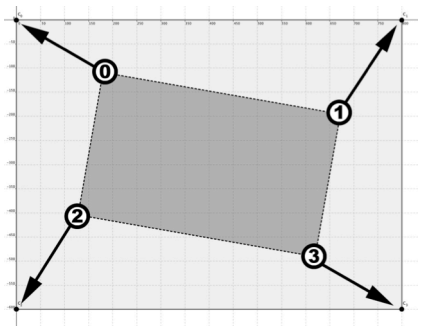
Figure 4a: The vertices of non-skewed images can be arranged in clockwise or counterclockwise order. For *V,* the order *V [0,2,3,1]* will always a yield correct arrangement.
Figure 4b: Skewed images, defined where *V* is skewed if *($V_0x < V_1x$ and $V_3x < V_2x$) or ($V_0x > V_1x$ and $V_3x > V_2x$)* must be ordered as *V [0,1,2,3]*.

is not an issue because the number of objects on the surface is rarely more than a dozen. In our implementation, both brute-force and quickhull algorithms were tested, with no observable difference in efficiency. In both cases, the software ran at greater than 30 frames per second.

Rendering the Game Board

Since the output in the Simple Configuration is simply to a graphics window on a computer monitor, rendering the game board is trivial using the facilities of the graphics library. Once the convex hull is determined, circles are drawn on the screen to represent all objects in the array, with exterior objects drawn in one color, and interior objects drawn in a contrasting color. Lines are between each object to clarify objects that lie within the boundaries of the hull.

Creating the game board in the Augmented Configuration is much more involved and creates an opportunity for working with geometric transformations. This challenge arises because the coordinates of game pieces in the images from the camera do not exactly match their positions in the image projected onto the game board (Figure 3b). This occurs because the camera and projector are slightly offset from each other and have different fields of



a.

```
beginShape();
  texture(V);                  \\ Input image
  vertex(180, 109, 0, 0);      \\ Vertex 0
  vertex(128, 404, 0, 600);  \\ Vertex 2
  vertex(620, 491, 800, 600);\\ Vertex 3
  vertex(672, 196, 800, 0);   \\ Vertex 1
endShape();
```

b.

Figure 5a: Process of mapping corners of a skewed section of an input image, *V[0,1,2,3],* to the projected image.
Figure 5b: Code for mapping *V* to the projected image with resolution 800 x 600 pixels.

view/projection. It is thus necessary to calibrate the area of interest in images from the camera, and adjust the projected output accordingly.

For our purposes, a simple four-point calibration suffices to provide accurate detection and projection of objects. An extra step is added, prior to initializing the reference frame for background subtraction, in which game pieces are placed on the four corners of the surface where projected image lies. The program identifies the coordinates of these corners within the image taken by the camera. Once these coordinates are determined, there are a couple of options for using them to map locations of detected objects in an input image to coordinates in the projected image. One option is to implement geometric transformations from scratch and apply them to the camera image. Alternatively, in Processing, a *texture()* function exists which uniformly maps any quadrilateral to another quadrilateral (Figure 4, Figure 5).

Rendering the outside edges of the convex hull can also present an interesting challenge. Given any set S{X, B, C, D, F, E, A} of exterior points in a convex hull, the exterior edges can be determined using a relatively simple geometric property of convex hulls. Any exterior point is chosen, and the angle between in and each of the remaining exterior points is calculated using some arbitrary interior point, Z. This will result in ∠XZB, ∠XZC, ∠XZD, ∠XZF, ∠XZE, ∠XZA (Figure 6). The order in which these angles are calculated is irrelevant. The angles are then ordered from least to greatest, ∠XZA, ∠XZB, ∠XZC, ∠XZD, ∠XZE, ∠XZF and their unique points extracted as an order of line segments to draw starting and ending with the originally chosen exterior point, i.e. segments, XA, AB, BC, CD, DE, EF and FX.

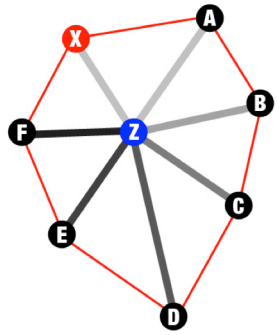


Figure 6: Rendering the exterior edges of a convex hull using the angles of the

REFLECTIONS ON GAME PLAY

As players throw their game pieces onto the board the software continually updates the convex hull and displays the results. The objective is to force an opponent's game piece on the board to become an interior point in the graph by landing one's piece at a point more exterior than theirs. Because pieces are thrown from a distance, the game requires some skill and one of five outcomes is possible from a throw:

1. The piece misses the game surface and is counted as a fault. Two faults results in a forfeit of the match.
2. The piece creates an exterior point of the graph and does not force any current piece to become an interior point. Game play continues.
3. The piece lands inside the convex hull as an interior point. The thrower loses the match.
4. The piece creates an exterior point and a new convex hull that has one or more of the opponent's pieces as an interior point. The thrower wins the match
5. The piece creates an exterior point and creates a new convex hull with a variety of interior points. The player with the most interior game pieces loses the match. In the event of an equal number of interior pieces, the thrower loses.

As play continues, it becomes increasingly difficult to create an exterior point and thus game play shifts to forcing a player's piece to become an interior point. An additional programming challenge is to automate the task of determining a winner identifying the losing

player according to the color or shade of the game pieces. After the outcome of the game is found, the winner remains while a new challenger replaces the loser and the game begins again.

While the Simple Configuration results in exciting game play, it is difficult for players to translate the image of the convex hull seen on the computer screen to what is occurring on the interactive surface. In the Augmented Configuration, however, the projector highlights pieces as they land and lines are rendered between each game piece so players can distinctly see where the next piece must land to create an exterior point. The addition of this feature has greatly added to the excitement of the game because it offers instant performance feedback to the players.

CONCLUSION

This paper has presented the development of a convex hull game with a tangible user interface as a context for engaging students in an Algorithms or Computer Graphics course. This exercise was effective in increasing the first author's understanding of convex hull and was later used to visualize other algorithms during the course. The game provides a spectrum of possibilities for implementation to fit a variety of skill sets while maintaining the goal of producing a fun, interactive, and visually impressive game that motivates the implementation of fundamental image processing and computer graphics techniques.

REFERENCES

[1] ACM CS2008 Curriculum Update, www.acm.org/education/curricula-recommendations, retrieved April 1, 2012.

[2] Cochran, Zane R. and Hamid, Nadeem Abdul. 2012. *Technical report: Convex hull game: a tangible context for algorithms and computer graphics concepts*, Mount Berry, Georgia: Berry College. Berry-CS-TR2012-1. 2 p. Available from http://cs.berry.edu/research.

[3] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., *Introduction to Algorithms (3rd ed.)*, Cambridge, MA: MIT Press, 2009.

[4] de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M., *Computational Geometry: Algorithms and Applications (3rd ed.)*, Berlin: Springer, 2008.

[5] Guzdial, M.J. and Ericson, B., *Introduction to Computing and Programming with Java: A Multimedia Approach*, Upper Saddle River, NJ: Prentice Hall, 2007.

[6] Ishii, H., Tangible bits: beyond pixels, *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, xv-xxv, 2008.

[7] Levitin, Anany V., *Introduction to the Design and Analysis of Algorithms (3rd ed.)*, Boston, MA: Addison-Wesley, 2012.

[8] Preparata, F.P. and Hong, S.J., Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM* 20, (2), 87-93, 1977.

[9] Processing.org, retrieved April 1, 2012.

[10] Yao, L., Dasgupta, S., Cheng, N., Spingarn-Koff, J., Rudakevych, O., and Ishii, H., RopePlus: bridging distances with social and kinesthetic rope games, *Proceedings of the 2011 Annual Conference Extended Abstracts on Human Factors in Computing Systems*, 223-32, 2011.