



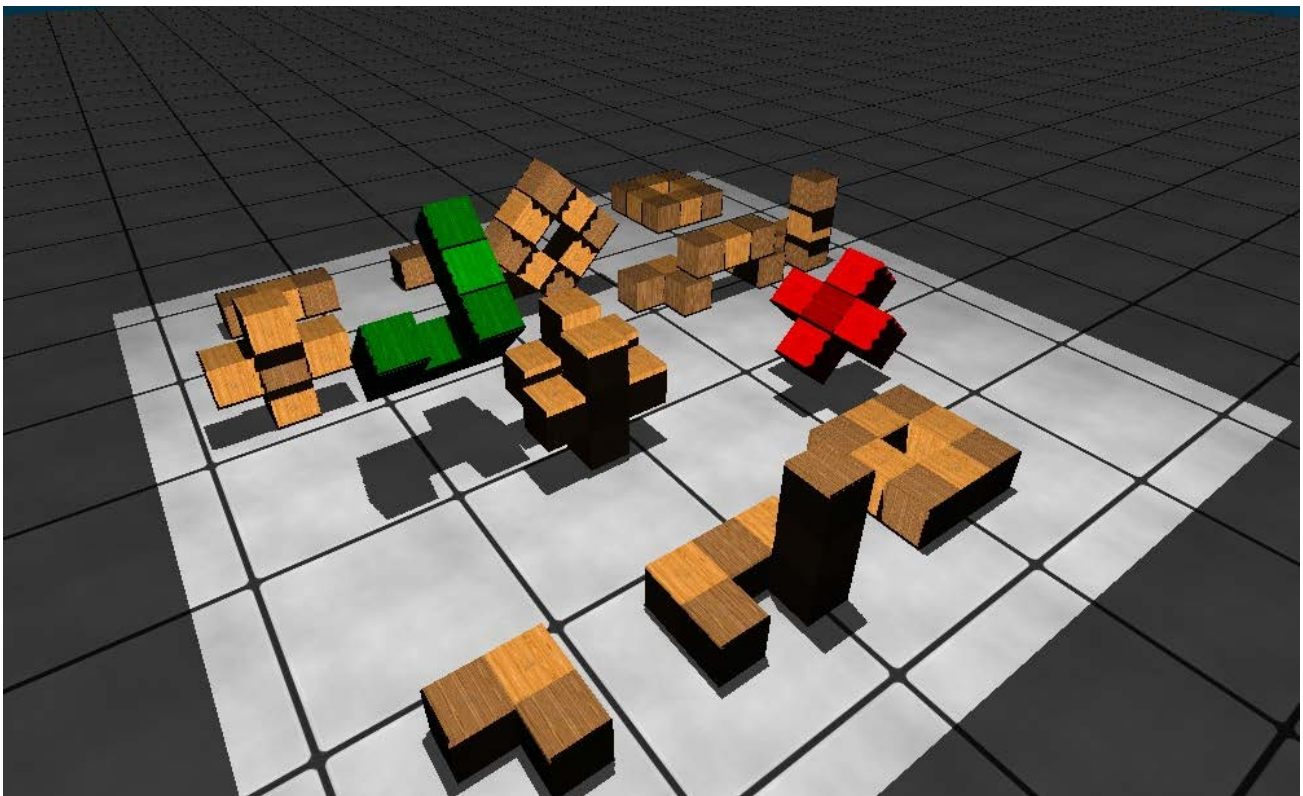
University of Peloponnese

Department of Computer Science and Technology

M. Sc. Program 2009 - 2011

Master thesis

Methods of Virtual Object Manipulation



Nikos Skiadas - pcst0929

Supervisor: Dr Georgios Lepouras

Table of contents

List of figures.....	05
Μέθοδοι διαχείρισης εικονικών αντικειμένων – Περίληψη.....	07
Abstract.....	11
Part 1 – The Problem.....	11
1) Introduction.....	11
2) Related work.....	12
2.1) The 3DUI 2011 contest.....	12
2.2) Other works on manipulation of virtual objects.....	13
Part 2 – Design.....	16
3) Bimanual object manipulation.....	16
3.1) Virtual hand utilization – Symmetry.....	17
3.1.1) Symmetrical hands.....	17
3.1.2) Non-Symmetrical Hands – use of dominant / non-dominant hand.....	17
3.2) Dragging vs. Grabbing.....	18
3.2.1) Dragging.....	18
3.2.2) Grab on/off.....	18
3.2.3) Comparison.....	18
3.3) Full Arm - Wrist mapping vs. Move – Rotate decoupling.....	18
4) Input devices.....	19
4.1) Immersive Input – Controllers.....	19
4.2) Desktop Controllers.....	22
4.3) Nonstandard Controllers.....	24
Part 3 – Implementation.....	25
5) Initial Approaches.....	25
5.1) Vizard VR Toolkit.....	25
5.1.1) Features.....	25
5.1.2) Why not?.....	25
5.2) Panda3D Game Engine	26
5.2.1) Features.....	26
5.2.2) Why not?.....	26
6) Environment – Software Libraries.....	27
7) Software Engineering & Classes.....	28
8) File Structure.....	29
8.1) Shape Library.....	29
8.2) Puzzle Description File.....	29
8.3) XML Configuration File.....	30
9) Rendering.....	31
9.1) Rendering Components.....	31
9.1.1) Cubes and Pieces.....	31
9.1.2) Cursors.....	33
9.1.3) Ground.....	33
9.1.4) Camera.....	34

9.2) Shadows.....	35
9.2.1) Method.....	35
9.2.2) Limitations and Issues.....	35
10) Physics Simulation.....	36
10.1) Limitations and Issues.....	37
11) Puzzle Recording and Evaluation.....	38
11.1) Relative transformations.....	39
11.2) Contact Points (cube faces).....	41
11.3) Grid matching.....	43
11.3.1) Determining the alternative solutions.....	44
Part 4 – Lessons Learned : Conclusions and Discussion.....	46
12) Simulation vs. user friendliness (or Is there too much physical representation?).....	46
13) Handedness and emergent usage.....	47
14) The Cursors : Form vs. Function.....	47
15) Viewing the world.....	48
16) “Natural” vs. Familiar.....	48
References.....	49

List of figures

Figure 1 : Playing HaptiCast.....	14
Figure 2 : Arx Fatalis spell casting with gestures and rune stones.....	14
Figure 3 : Laboratory analogy.....	16
Figure 4: The Nintendo Wii Remote.....	19
Figure 5: The Microsoft Kinect.....	20
Figure 6: The PlayStation Move.....	21
Figure 7: Common desktop mouse.....	22
Figure 8: The 3DConnexion CADman and Space Pilot 3D mice.....	23
Figure 9: The SensAble Phantom Omni device.....	23
Figure 10: Color Glove tracking system.....	24
Figure 11: The PCubee 3D display device.....	24
Figure 12: A real-world cube lock puzzle and a rendering of a set of pieces in Puzzle3D.....	32
Figure 13: Tiled vs. Seamless texture map on the ground.....	33
Figure 14 : Horizontal Coordinates.....	35
Figure 15: An example of relative positioning equivalence.....	40
Figure 16: An 2D representation of a composite shape with hull normals.....	41
Figure 17: An example of contact point matching equivalence.....	42
Figure 18: Puzzle solution evaluation with grid matching.....	43
Figure 19: Constructing alternative puzzle solutions.....	45

Μέθοδοι διαχείρισης εικονικών αντικειμένων

Περίληψη

Σε πολλά παραδείγματα διάδρασης ανθρώπου – υπολογιστή, η επικοινωνία με το σύστημα περιλαμβάνει απ' ευθείας διαχείριση εικονικών αντιστοιχών πραγματικών (φυσικών) αντικειμένων. Καθώς τα υπολογιστικά συστήματα εξελίσσονται επιτρέποντας όλο και πιο εξελιγμένες συσκευές εισόδου που προσφέρουν ακριβέστερο έλεγχο και μεγαλύτερη ελευθερία, και η απεικόνιση γραφικών πλησιάζει βαθμιαία το φωτορεαλισμό, εμφανίζονται όλο και περισσότερες εφαρμογές εικονικής σύνθεσης αντικειμένων (κατασκευής). Στην εργασία αυτή παρουσιάζουμε μια εφαρμογή που επιτρέπει στο χρήστη να «λύσει» έναν απλό κατασκευαστικό 3D γρίφο (ruzzle) αλληλεπιδρώντας απ' ευθείας με τα κομμάτια που τον απαρτίζουν σε 3D χώρο και σε πραγματικό χρόνο, και αναλύουμε τις μεθόδους που χρησιμοποιήθηκαν καθώς και τα συμπεράσματα και τις διαπιστώσεις που προέκυψαν κατά την υλοποίηση.

Καθώς δεν υπάρχει περιορισμός ως προς τον τύπο των αντικειμένων που μπορούν να αναπαρασταθούν σε ένα εικονικό περιβάλλον, τα εικονικά αντικείμενα μπορεί να είναι αναπαραστάσεις αντικειμένων που δεν είναι «διαχειρίσιμα» στον πραγματικό κόσμο (αντικείμενα μεγάλου όγκου / βάρους) ή φανταστικών ή αφηρημένων αντικειμένων. Στην παρούσα εργασία, ο όρος «εικονικό» αναφέρεται στην αναπαράσταση αντικειμένων με φυσικά αντίστοιχα σε εικονικό (υπολογιστικό) περιβάλλον.

Με βάση αυτή τη θεώρηση γίνεται αναγκαία η αναπαράσταση των αντικειμένων σε περισσότερες «διαστάσεις» από την απλή οπτική αναπαράσταση: η διαχείριση των αντικειμένων εισάγει ζητήματα όπως η αναπαράσταση του βάρους, του ενεργού όγκου (των διαστάσεων του αντικειμένου ώστε να πραγματοποιούνται ρεαλιστικές αλληλεπιδράσεις) κτλ.

Στην υλοποίησή μας προσπαθήσαμε να παράγουμε μια επαρκή αναπαράσταση των αντικειμένων σαν φυσικές οντότητες με χαρακτηριστικά όπως βάρος, αλληλεπίδραση δυνάμεων και τριβές σε ένα δυναμικό κόσμο.

Οι προδιαγραφές και ο σχεδιασμός της εργασίας επηρεάστηκε κυρίως από την περιγραφή του διαγωνισμού 3DUI 2011.

Μια βασική επιδίωξη της εργασίας είναι να μελετήσουμε μεθόδους διάδρασης κατά τις οποίες εμπλέκονται και τα δύο χέρια του χρήστη. Επιδιώξαμε να μεταφέρουμε την αίσθηση ότι ο χρήστης έχει τα κομμάτια στη διάθεσή του σε ένα εικονικό «τραπέζι» και μπορεί να αλληλεπιδράσει με αυτά με τον τρόπο που θα χρησιμοποιούσε τα χέρια του. Κατ' αυτή την έννοια δεν υπάρχει αναπαράσταση του χρήστη σαν κάποια ανθρωπομορφική οντότητα στο χώρο του ruzzle, αλλά μόνο η αναπαράσταση των αλληλεπιδραστικών του «μέσων» (δύο σφαίρες που αναπαριστούν τη θέση του κάθε χεριού στο χώρο).

Εξετάζουμε τις περιπτώσεις συμμετρικής και μη συμμετρικής χρήσης των χεριών: στην πρώτη περίπτωση τα εικονικά χέρια του χρήστη είναι ισοδύναμα ως προς τη λειτουργικότητα και τον τρόπο που αυτή αποδίδεται με τις διαθέσιμες συσκευές εισόδου, ενώ στην δεύτερη υπάρχει διάκριση κυρίαρχου / μη κυρίαρχου (dominant / non dominant hand) για εξειδικευμένη λειτουργικότητα κατά περίπτωση η οποία υποστηρίζεται και με τη χρήση διαφορετικών συσκευών. Επίσης αναλύουμε τις διαθέσιμες επιλογές που αφορούν τον τρόπο με τον οποίο ο χρήστης κρατά και διαχειρίζεται ένα κομμάτι καθώς και τον τρόπο με τον οποίο αποδίδεται η κίνηση του χεριού.

Κάνουμε μια σύντομη παρουσίαση των μονάδων εισόδου που λάβαμε υπ' όψη κατά το σχεδιασμό και παρουσιάζουμε τα χαρακτηριστικά τους, καθώς και μια ομαδοποίησή τους ως προς τον τύπο αλληλεπίδρασης που προϋποθέτουν και τη σχέση χρήστη – κίνησης – χώρου που επιβάλλουν. Πιο συγκεκριμένα γίνεται ξεχωριστά λόγος για συσκευές «εμβύθισης» (immersive), επιτραπέζιες συσκευές και κάποιες χαρακτηριστικές πειραματικές (με την έννοια «μη καταναλωτικές») συσκευές.

Περιγράφουμε αναλυτικά την υλοποίηση και τις μεθόδους που ακολουθήσαμε. Αρχικά γίνεται μια αποτίμηση αρχικών προσεγγίσεων (APIs, εργαλεία ανάπτυξης) και καταγράφονται οι λόγοι για τους οποίους δεν επιλέξαμε τις συγκεκριμένες λύσεις. Αναφέρονται οι βιβλιοθήκες λογισμικού που χρησιμοποιήθηκαν στην υλοποίηση (η οποία έγινε σε γλώσσα C / C++) που παρουσιάζεται, και τα βασικά τους χαρακτηριστικά. Επίσης αναφέρονται οι βασικότεροι τύποι δεδομένων (ADTs) που δημιουργήθηκαν και γίνεται σύντομη δομική περιγραφή της εφαρμογής.

Καθώς μια από τις κυριότερες απαιτήσεις της εφαρμογής ήταν να μπορούμε να ορίζουμε με απλό τρόπο νέους τύπους κομματιών του ruzzle αλλά και νέους σχηματισμούς (γρίφους) προς λύση, παρουσιάζουμε τη μέθοδο που ακολουθήσαμε και τη μορφή των αρχείων που χρησιμοποιεί η εφαρμογή για να παράγει δυναμικά (κατά την εκτέλεση) νέα είδη κομματιών με συνεπή «φυσική» συμπεριφορά. Πιο συγκεκριμένα, ο μορφολογικός περιορισμός των κομματιών είναι ότι μπορούμε να θεωρήσουμε πως αποτελούνται από ένα σύνολο ομοιόμορφων κύβων, οι οποίοι «συγκολλούνται» για να παράγουν το κάθε σχήμα. Κατ' αυτό τον τρόπο, το αρχείο περιγραφής αποθηκεύει τις θέσεις των κύβων που παράγουν το κάθε κομμάτι σαν ακέραιες συντεταγμένες σε χώρο αντικειμένου – model space (και όχι σε χώρο κόσμου – world space).

Στη συνέχεια αναλύονται οι επιλογές που κάναμε σχετικά με την απεικόνιση του χώρου του ruzzle. Οι επιλογές αυτές αφορούν:

- Την απεικόνιση των κύβων και των κομματιών, και «φυσική» τους εμφάνιση
- Την απεικόνιση των «δρομέων» (cursors) που αναπαριστούν τα χέρια του χρήστη και τη λογική με βάση την οποία σχεδιάστηκαν
- Την απεικόνιση του εικονικού μας «τραπεζιού» - εδάφους και τη χρήση του για την καλύτερη απόδοση αίσθησης βάθους, προοπτικής και σχετικής θέσης των κομματιών
- Τη διαχείριση της εικονικής κάμερας, ή αλλιώς τη δυνατότητα ελέγχου του σημείου παρατήρησης που δίνεται στο χρήστη
- Τη μέθοδο απεικόνισης σκιών καθώς τα οφέλη που παρουσιάζει αυτή στην καλύτερη αντίληψη θέσης και απόστασης των κομματιών

Επεξηγούνται η χρήση της μηχανής προσομοίωσης φυσικής, και ο τρόπος που τα κομμάτια του ruzzle αποκτούν υπόσταση στον χώρο του που πραγματοποιείται η διάδραση. Επίσης οι μέθοδοι με τις οποίες το σύστημα «αντιλαμβάνεται» τις πράξεις του χρήστη (πώς ανιχνεύεται το ότι «αγγίζουμε» ένα κομμάτι, πως πιάνουμε ή αφήνουμε ένα κομμάτι). Τέλος γίνεται αναφορά στους περιορισμούς που επιβάλλει η μηχανή προσομοίωσης ως προς την ακρίβεια, ή θέματα που έχουν να κάνουν με την ποιότητα της αλληλεπίδρασης και περιγράφονται οι λύσεις που υιοθετήσαμε. Σε κάποιες περιπτώσεις «αποκλείσαμε» κάποια χαρακτηριστικά της προσομοίωσης με αποτέλεσμα να μοιωθεί ο ρεαλισμός του τελικού αποτελέσματος, κερδίζοντας όμως έτσι σε ευχρηστία. Αυτό είναι και ένα από τα θέματα που διερευνώνται στα συμπεράσματα.

Κλείνοντας με τα θέματα υλοποίησης, παρουσιάζονται οι λύσεις που εξετάσαμε για το θέμα της καταγραφής της ορθής λύσης του γρίφου (κατά την κατάσταση «προγραμματισμού» του), και της αξιολόγησης της λύσης κατά την κατάσταση παιχνιδιού. Παρουσιάζονται 3 προσεγγίσεις, τα προβλήματα που παρουσίασαν και οι περιορισμοί τους και τελικά γιατί καταλήξαμε στη λύση που υλοποιήθηκε απορρίπτοντας τις 2 άλλες.

Ειδικότερα παρουσιάζονται 3 μέθοδοι καταγραφής:

- Καταγραφή των σχετικών θέσεων κάθε ζεύγους κομματιών
- Καταγραφή των σημείων επαφής των ελεύθερων επιφανειών κομματιών
- Δημιουργία ενός περιέχοντος 3D κάναβου (occupancy grid) για τους κύβους που αποτελούν τα κομμάτια – λύση που προτιμήθηκε

Τέλος αναλύονται τα συμπεράσματα που προέκυψαν γύρω από τους εξής θεματικούς άξονες:

- Ποιότητα προσομοίωσης και ρεαλισμός σε σχέση με ευχρηστία και ευκολία χειρισμού
- Αλληλεπίδραση με τα δύο χέρια και πως αυτή προκύπτει σε ένα περιβάλλον ανοιχτό σε προσαρμογή / εξέλιξη του χρήστη
- Η αφαιρετική αναπαράσταση των χεριών του χρήστη και τα οφέλη της
- Η δυνατότητα του χρήστη να αλλάζει την οπτική γωνία του παιχνιδιού και η σημασία της στη δυνατότητα χειρισμού των κομματιών και της κατασκευής
- Η σχέση που έχει η οικειότητα του χρήστη με μια συσκευή εισόδου με την ευκολία χειρισμού. Η αντίθεση εδώ είναι ανάμεσα σε μια γνωστή αλλά πιθανόν λιγότερο κατάλληλη για το συγκεκριμένο σενάριο συσκευή, σε σχέση με μια πληρέστερη λειτουργικά αλλά «άγνωστη» συσκευή.

Abstract

In many examples of human-computer interaction, interfacing with the system involves direct manipulation of virtual counterparts of physical objects. As computer systems evolve allowing increasingly sophisticated input devices with finer control and greater freedom, and graphics display is gradually reaching photo – realism, virtual assembly applications are not uncommon. We present an application that allows the user to solve a simple 3D Puzzle by directly interacting with the building blocks (pieces) in 3D space, and discuss the methods used as well as the insights gained in the course of implementation.

Part 1 – The Problem

1) Introduction

In this work we implement an application that allows solving a simple 3D puzzle by directly interacting with the puzzle pieces. Our aim is to evaluate different approaches and identify at least some of the problems that arise when trying to provide the user with a method of direct interaction with a compound virtual object.

As there is no limitation to the type of objects that can be represented in a virtual environment, virtual objects can be representations of items that could not be physically manipulated in the real world (for example immobile objects or objects of excessive size / weight) or entirely fictional or abstract objects. In this study however, we are mainly interested in items with physical counterparts, with well-known or obvious affordances. As such the term “virtual” here refers to virtual representations of physical items in a virtual environment.

In the latter case there are certain aspects of the representation of a virtual item that should be taken into account. Apart from the visual representation of an object, direct manipulation also introduces issues of representing the “weight” of an object – at least in a relative fashion to the weight of other objects of the same virtual world, the volume of the object, which of course refers not only to the volume perceived by the user through graphically rendering the object, but also the interactive volume of the object – meaning collision space, interaction with other objects (such as joining, breaking) and so on.

In the present implementation, we tried to give an adequate representation of the game objects as physical items in a (restricted) dynamic world, having properties like weight, friction and gravity, and presenting natural behavior (collisions and force response). The aspect of force feedback that would provide the user with additional tactile data has been left for further study (see also §10.1).

2) Related work

2.1) The 3DUI 2011 contest

The description and specifications of the present work, were mainly inspired by the call to the 3DUI 2011 contest [1]. The terms of the content hint rather than explicitly specify that the goal of the interaction is to assemble a puzzle given its building blocks, and not to find a way to disassemble it in a “Cube Lock” puzzle fashion. We considered the two cases to be symmetrical in terms of the way the user manipulates the puzzle building blocks, and as such no special conclusions would be reached by favoring either case. Our implementation presents the user with a set of “free” pieces, the goal being to assemble the puzzle.

Video presentations of the results of the contest have been available:

Felipe Bacim, Cheryl Stinson, Bireswar Laha and Doug Bowman (The Fighting Gobblers team, Department of Computer Science and Center for Human Computer Interaction, Virginia Tech) present a method to not only build a puzzle but essentially build the pieces in an interactive manner, by selecting “occupation” space in a fixed grid [19]. Their approach uses a custom immersive controller (for the term see §2.5.1, “Immersive Input – Controllers”) to manipulate the puzzle space itself, as well as make selections by pointing in an on – screen menu.

Billy Lam, Yichen Tang, Ian Stavness and Sidney Fels (Electrical and Computer Engineering, University of British Columbia) use their pCube device [27], [28] to allow the user to have direct control over the puzzle 3D space view [20]. pCube is a multi – faceted display that credibly creates the illusion of holding a transparent box with real contents.

Steven Maesen, Patrik Goorts, Lode Venacken, Sofie Notelaers and Tom De Weyner (Hasselt University) presented an augmented reality environment that allows direct manipulation of real-world objects. The system tracks the motion of the “controller” real objects, and translates it to translations and rotations of the virtual ones [21]. This has the advantage of immediate tactile feedback (volume, weight) while the augmented reality display presents the state of the virtual puzzle world to the user. Of course, here as in any augmented reality system, the coupling of virtual and real objects is purely artificial – what the user “perceives” she/he is handling, does not necessarily coincide with the item at hand.

Tuukka Takala and Roberto Pugliese (Aalto University, Department of Media Technology) presented a solution based on their CAVE – like immersive environment RUIS (Reality – based User Interface System), that also uses Wiimote and similar controllers [22].

Dmitri Shuralyov and Wolfgang Stuerzlinger (Department of Computer Science and Engineering, York University) presented a more “conventional” solution, using standard desktop equipment, and providing object manipulation that resembles a 3D modeling application environment. [23]

The contest entry by Masashi Kitagawa and Tsuyoshi Yamamoto (Hokkaido University) is not a virtual environment that allows the user to interact with virtual objects to solve or construct a puzzle, but rather an augmented reality system that offers a step by step guidance to solving a real – world puzzle with actual pieces [24]. Obviously the system must know the puzzle structure, and be able to recognize that a certain step has been carried out. The display system combines a tracker with a projector in order to monitor the piece positions and display instructions and information on the puzzle space plane (piece numbers, pointers and schematics)

Toni Da Luz presents a method to manipulate a cursor (actually an action 'space' displayed as a box) in order to carry out all interactions with puzzle pieces (selecting, moving rotating and placing) [26] using the novel multi – touch device Cubtile [25] by Immersion SAS.

Henrique Debarba, Juliano Franz, Vitor Reus, Anderson Maciel and Luciana Nedel (Instituto de Informatica, Universidade Federal do Rio Grande do Sul) demonstrate a very interesting combination of input controllers: in this proposal, there is explicit distinction of dominant / non dominant hand, in the form of greater / lesser demand for movement accuracy respectively. The dominant hand is responsible for single piece manipulation using a Phantom Omni device (see 2.5.2 Desktop Controllers) while the non – dominant hand alters the orientation of the puzzle space using data gathered by tracking an iPod smart phone. The implementation also incorporates haptic feedback through the Phantom Omni, view (camera) freedom through head tracking, and stereoscopic vision. [30] Our own take on hand differentiation is discussed in (2.2, “Virtual hand utilization – Symmetry”)

2.2) Other works on virtual object manipulation

Direct manipulation of virtual objects in a virtual environment, spans a wide range of issues: input methods, means of registering the user's actions in the system, object interactions and behaviors in the system world, and so forth.

Yong Wang, and Sankar and Uma Jayaram, present some aspects of representing the objects behaviors in a physically correct manner [2]. Except from providing with a credible natural representation (which we have “bypassed” in some cases, as discussed in §4.1) the purpose of using a real-time physics simulation engine in our implementation, was to address the type of issues presented here.

Lee, Billingham, and Woo have developed an augmented reality method of bimanual user input through a custom built pair of tangible cubes that support the screw – driving and block assembly methods they propose [5]. The user interacts directly with the cubes, while a set of enclosed magnets can track their relative position and motion in real space, and translate the data accordingly to detect the user's movements that indicate intent to fit together or decouple the objects at hand.

Andrews, Mora, Lang and Lee present an implementation of object manipulation in a gaming context (using the SensAble Phantom Omni device presented in §4.2) in the form of HaptiCast [3]. Quoting from the game description “*In HaptiCast, players take on the role of a wizard, with a set of haptically – enabled wands to help them do battle and interact with their environment*”. In the context of HaptiCast, the issue of dominant and non – dominant hand is apparent.



Figure 1 : Playing HaptiCast

Another attempt to convey the avatar's physical presence in a game is the 2002 first person Role-Playing game *Arx Fatalis* by Arkane Studios [4]. In *Arx Fatalis* the player, besides fighting with “traditional” pseudo – medieval weapons, can cast magical spells in an interesting fashion: in order to cast a spell she/he must first discover the appropriate runic stones which depict a sequence of gestures in the form of an abstract shape, and then “draw” the shape using mouse movement, mapping in-game to the avatar's hand.



Figure 2: Arx Fatalis spell casting with gestures and rune stones

Kiyokawa, Takemura, Katayama, Iwasa and Yokoya have implemented the VLEGO system [10], a prototype immersive modeler that employs features of toy blocks to give flexible two – handed interaction for 3D design. In this work many interesting issues about block collision states are discussed. Also, VLEGO highlights the need to make location and orientation of primitives discrete.

Brandl, Forlines, Wigdor, Haller and Shen [33] discuss the benefits of differentiating and combining dominant and non – dominant hand in the context of bimanual pen and direct – touch interaction on horizontal (drawing) surfaces.

Hinckley, Czerwinski and Sinclair present a similar case of differentiation between hands [9] using a touchpad for the non – dominant hand and a customized mouse for the dominant hand, and show detailed transition diagrams for hand states during interaction.

Kunert, Kulik, Huckauf and Fröhlich compare tracking- and controller- based input for complex bimanual interaction in virtual environments in [8]. One of the results of interest to the present work is that maneuvering could be performed easier with a controller – based device.

Murayama, Bougrila, Luo, Akahane, Hasegawa, Hirsbrunner and Sato present SPIDAR – G [6], a novel haptic 6DOF interface for bimanual VR interaction. SPIDAR provides effective force and torque feedback by design.

Ott, Thalmann and Vexo discuss the benefits and difficulties in using mixed – reality environments for industrial training [34], using an augmented reality implementation with force feedback

Manipulation of virtual objects is very important in cases where the physical counterparts are not necessarily available, or too risky to handle. Such a case is manipulation of objects of cultural heritage for purposes of reassembly efforts. Solutions specifically targeted to this purpose are presented in [35] and [36]. The term “solutions” here reflects the fact that the applications presented do not rely solely on software approaches, but also introduce custom devices that (in the case of ArcheoTUI) even involve foot pedals. A more specific discussion concerning modality relative to the ArcheoTUI environment can be found in [38].

Adams, Klowden and Hannaford [37] conduct an experiment in training users to perform a manual task through a force feedback enabled virtual environment, and discuss the results.

Schlattmann and Klein discuss the issues arising from using directly tracking the user's hands to provide for markerless 3D interaction. One of the main problems in this context is that there is no direct way to indicate a “switch”, for example a grab on – off button (see also §3.2.2), so the relative positioning of hands is used to indicate “grab”, “release” and so on.

Part 2 – Design

3) Bimanual object manipulation

For our implementation we chose not to represent the user as an anthropomorphic entity in the puzzle space. Focus of the application is what the user can do with his hands, provide adequate response to the hand movement, and restrict his interaction with the available puzzle building blocks, in order to provide a form of “confined sandbox”.

The omission of some kind of avatar for the user

(a) Removes the burden of controlling the avatar, providing for a more direct experience for the user (as if she/he holds the items in his own hands)

(b) Simplifies the input method and allows for every degree of freedom provided by the controllers to be mapped to hand movements and

(c) Removes the visual obstruction of a redundant entity – that would be the avatar model - besides the puzzle pieces and cursors from the puzzle space.

In a real world analogy, the intention was to provide freedom in a confined environment. Figure 2 depicts an instance of laboratory handling of radioactive materials, as an analogy of our view of the puzzle space.



Figure 3 : Laboratory analogy

3.1) Virtual hand utilization – Symmetry

Virtual hand utilization refers to the manner in which the user's hands representations in the puzzle space are enabled or limited. It has nothing to do with the physical ability of the user to interact with the provided controllers. This said, we considered two main “paradigms”:

3.1.1) Symmetrical hands

User “hands” are equivalent in terms of ability to manipulate pieces as well as their binding / response to input devices. This approach implies use of similarly equivalent input devices.

Advantages

- User does not have to provide direct or indirect information about his own handedness – which in addition would be considered intrusive – and is allowed to evolve his interaction with the system as she/he sees convenient.
- There are no “special cases” considering the implementation, both virtual hands are considered equal in terms of system functionality

3.1.2) Non-Symmetrical Hands – use of dominant / non-dominant hand

User defines a dominant and a non – dominant hand (potentially by setting a “handedness” option). We considered this option in the sense that usually we perform tasks with greater precision with our dominant hand, and this could be mapped by assigning it a controller with greater granularity / more degrees of freedom.

Advantages

- We tend to automatically assign different tasks to each hand when dealing with bi-manual operations. Input device differentiation naturally maps this innate tendency.
- Better utilization of controller's response / granularity.

Disadvantages

- Differentiation of user's hands is artificial / enforced → user must setup the controller to be used by each hand based on a priori knowledge of what suits him.
- Unless we provide with a method to identify the dominant hand implicitly, we force the user to explicitly present a characteristic that should be naturally transparent in a real – world scenario.

3.2) Dragging vs. Grabbing

Considering the state on “holding” a virtual object in each of the virtual “hands”, we discussed two options:

3.2.1) Dragging

When the system registers required proximity (“touching”) between a hand representation in the virtual world (a cursor) and a puzzle piece, the user can press a controller switch to indicate that she/he seizes the piece. In order to keep the piece at hand, she/ he must keep the switch suppressed. Releasing the switch corresponds to releasing the grip to the piece, so the cursor / hand is again considered empty.

3.2.2) Grab on/off

When the state described above is detected, the user presses and releases a controller switch to indicate that the piece is actively in the corresponding hand (cursor). The piece remains controlled by the grabbing cursor until the user presses and releases the same switch.

3.2.3) Comparison

Although the dragging method seems to map more naturally to the notion of “holding” an item, early tests indicated that keeping a switch pressed can be a burden to the users hands, and makes the use of the controller more complex and – interestingly – more unnatural. Besides, the percentage of time the user is holding an item, is expected to be far greater the one spent with empty cursors, so taxing the user with one more button to press is considered unnecessary. This option however, might have some sense coupled with using force feedback in order to provide a “full” tactile experience (sense of weight / pressure) while carrying an item. As a result we decided in favor of the grabbing option for its greater ease of use.

We can argue that for the 3D mouse, where there is not button on the cap controlling motion and manipulating a 3D object requires precise movements (possibly taking longer), dragging was considered difficult for users. In the case of a joystick or a haptic device such as Phantom Omni it may have been easier (especially if manipulation movements are short).

3.3) Full Arm - Wrist mapping vs. Move – Rotate decoupling

Sophisticated input devices such as the 3Dmouse tested offer 6 degrees of freedom (represented with registering movement in all axes) that can potentially map the full spectrum of the movement of a human hand. For the purpose of this work we can recognize two major forms of hand motion:

- arm motion which maps to translating an active item in the puzzle world and
- wrist motion which maps to rotating an active item in the puzzle world around its current world position

The immediately obvious approach was to try and give the user the full range of hand motion by mapping the whole axis set provided by the controller to item rotation and translation simultaneously. User testing, initially gave the impression that this is counter intuitive in the case of 3Dmouse.

Arm motion and wrist motion require quite different activation of our motor facilities and provide a very different feedback. Arm motion is targeted to longer range movement, fast and with no particular precision requirements (though we can perform very accurately) while wrist motion is targeted to working with finer

control over smaller items at hand, and allows for very precise movement. Given an input device such as a 3Dmouse, the user is expected to map these different types of motion to one device that provides the same feedback for every type of movement (the user receives the same “impedance” from the controller for every axis), and use a limited range of movement (essentially only wrist movement) for all actions.

In the course of implementation, we considered the solution of adding a switch to indicate that the controller was in “piece translate” or “piece rotate” mode. This solution however added another artificial barrier to the actual translation of the user's movements and consequently lessened the experience of direct piece manipulation (as was expected by adding a new mode in user interaction).

After performing some more tests, we concluded that the solution lied simply on carefully adjusting the controller's input to a more suitable translation / rotation speed and adding some thresholds to alleviate the controller's sensitivity, in order to to match the expected accuracy. The result was that the “collateral” rotation during translation or vice – versa was within acceptable limits, and user experience became more smooth and predictable.

4) Input devices

In this section we briefly present some of the options for input controllers available today and their primary attributes. The majority of these devices are market standards in one way or another, however, we also present a few experimental methods of user input.

4.1) Immersive Input – Controllers

We use the term immersive here to describe the type of controllers that take into account the user's spatial interaction properties, such as bodily movement, posture, and speed. This kind of input devices is rapidly becoming dominant in home entertainment appliances, and actually, the great majority of applications that take advantage of this technology is video games.



- Motion sensing (vertical axis, left – right horizontal axis, horizontal rotation)
- infrared optical sensor
- 8xDigital buttons (A, B, +, -, Home, 1, 2, Power)
- Digital D – pad

The Wii remote is the primary controller for the console. It uses a combination of built-in accelerometers and infrared detection to sense its position in 3D space when pointed at the LEDs within the sensor bar. This design allows users to control the game using physical gestures as well as traditional button presses. The controller connects to the console using Bluetooth and features rumble as well as an internal speaker.

Source : <http://en.wikipedia.org/wiki/Wii>

Microsoft Kinect

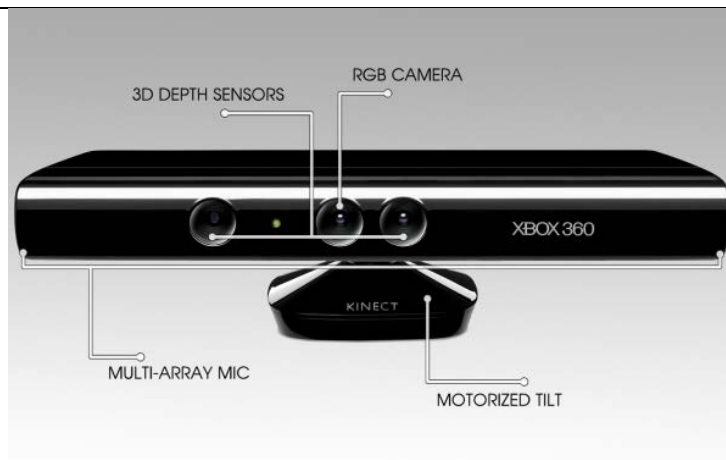


Figure 5: The Microsoft Kinect

Sensor

- Color and depth-sensing lenses
- Voice microphone array
- Tilt motor for sensor adjustment

Field of View

- Horizontal field of view: 57 degrees
- Vertical field of view: 43 degrees
- Physical tilt range: ± 27 degrees
- Depth sensor range: 1.2m – 3.5m

Data Streams

- 320x240 16 bit \rightarrow 30 fps
- 640x480 32 bit \rightarrow 30 fps
- 16 bit audio 16 KHz
- Skeletal tracking system
- Up to 6 people, 2 active players
- 20 joints per active player
- Can map active players to live avatars

Audio

- LIVE party chat and in-game voice chat
- (requires Xbox LIVE Gold Membership)

- Echo cancellation system
- Multiple speech recognition

- The RGB camera takes pictures and sends via USB to the computer.
- The Depth sensors are used to measure the 3rd dimension that is the Depth of the object from the camera.
- The Microphones are used to take the Audio input to the computer via USB.
- The Motor and Tilt mechanism is used in the case some object in front of the camera is to be tracked in its motion





Source : <http://entreprene.us/2011/03/09/microsoft-kinect-technical-introduction/>

PlayStation Move





Figure 6: The PlayStation Move

Motion controller

- Motion sensing (three-axis accelerometer, three-axis angular rate sensor)
- Location tracking (magnetometer, object recognition (via PlayStation Eye))
- 1 Analog trigger (T)
- 8 buttons (, , , , Start, Select, Home, Move)

Navigation controller

- Analog stick
- D-Pad
- 1 Analog trigger (L2)
- 5 buttons (, , L1, Home, L3)

The PlayStation Move motion controller features an orb at the head which can glow in any of a full range of colors using RGB light-emitting diodes (LEDs). Based on the colors in the user environment captured by the

PlayStation Eye camera, the system dynamically selects an orb color that can be distinguished from the rest of the scene. The colored light serves as an active marker, the position of which can be tracked along the image plane by the PlayStation Eye. The uniform spherical shape and known size of the light also allows the system to simply determine the controller's distance from the PlayStation Eye through the light's image size, thus enabling the controller's position to be tracked in three dimensions with high precision and accuracy. The sphere-based distance calculation allows the controller to operate with minimal processing latency, as opposed to other camera-based control techniques on the PlayStation 3. A pair of inertial sensors inside the controller, a three-axis linear accelerometer and a three-axis angular rate sensor, are used to track rotation as well as overall motion. An internal magnetometer is also used for calibrating the controller's orientation against the Earth's magnetic field to help correct against cumulative error (drift) by the inertial sensors. The inertial sensors can be used for dead reckoning in cases which the camera tracking is insufficient, such as when the controller is obscured behind the player's back.

Source : http://en.wikipedia.org/wiki/PlayStation_Move

4.2) Desktop Controllers

Common desktop mouse



Figure 7: Common desktop mouse

Implementation variants

- Mechanical mice
- Optical and Laser mice
- Inertial and gyroscopic mice
- 3D mice
- Tactile mice

A mouse is a pointing device that functions by detecting two – dimensional motion relative to its supporting surface. Physically, a mouse consists of an object held under one of the user's hands, with one or more buttons. It sometimes features other elements, such as "wheels", which allow the user to perform various system-dependent operations, or extra buttons or features that can add more control or dimensional input. The mouse's motion typically translates into the motion of a cursor on a display, which allows for fine control of a graphical user interface.

Source : [http://en.wikipedia.org/wiki/Mouse_\(computing\)](http://en.wikipedia.org/wiki/Mouse_(computing))

3D Mouse (input method used)



Figure 8: The 3DConnexion CADman and Space Pilot 3D mice

- 6DoF devices for quickly orienting 3D objects or views
- Devices that enable working with both hands simultaneously (for example, a 3D mouse in one hand and a traditional 2D mouse in the other hand)

Commonly utilized in CAD applications, 3D modeling, animation, 3D visualization and product visualization, users can manipulate the controller's pressure-sensitive handle (historically referred to as either a cap, ball, mouse or knob) to fly through 3D environments or manipulate 3D models within an application. The appeal of these devices over a mouse and keyboard is the ability to pan, zoom and rotate 3D imagery simultaneously, without stopping to change directions using keyboard shortcuts or a software interface

Source : <http://en.wikipedia.org/wiki/3DConnexion>,
http://www.3dconnexion.com/fileadmin/user_upload/manuals_docs/english_intl/3dx_whitepaper_cadpayback_en_intl.pdf

SensAble Phantom Omni



Figure 9: The SensAble Phantom Omni device

- 6 degree – of – freedom positional sensing
- Two integrated momentary switches on the stylus for ease of use and end – user customization

- Stylus – docking inkwell for automatic workspace calibration

The SensAble Phantom Omni device is a 6 DoF desktop input device that can represent 3D rotations and translations as well as provide application defined force feedback through a series of motors. This feature makes for good use in physics simulation – enabled environments, as it can convincingly convey the sense of weight, resistance and friction.

Source : <http://www.sensable.com/haptic-phantom-omni.htm#techspecs>

4.3) Nonstandard Controllers

Color glove



Figure 10: (Left, Middle) Color Glove tracking system, (Right) A 3D rendering of a Color Glove interaction environment

Quoting from the paper : “Our approach uses a single camera to track a hand wearing an ordinary cloth glove that is imprinted with a custom pattern. The pattern is designed to simplify the pose estimation problem, allowing us to employ a nearest-neighbor approach to track hands at interactive rates”

See [11]

PCubee

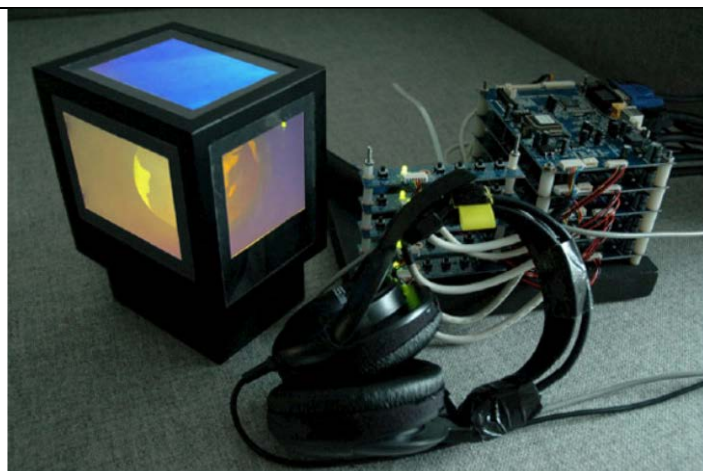


Figure 11: The PCubee 3D display device

Quoting from the paper : “The pCubee display uses five 5” LCD panels arranged as five sides of a box. Three

graphics pipelines drive the screens on the sides of the box visible to the user. A Polhemus Fastrak is used to track the pCubee and users' head in order to couple the user's view of the box to the rendered perspective on each screen. This creates the illusion for the user of looking into a box with clear sides to see a small virtual diorama contained within."

See [27], [28], [29]

Part 3 – Implementation

5) Initial Approaches

Before deciding in favor of the current implementation, we experimented with a few other alternatives considering authoring tools and APIs

5.1) Vizard VR Toolkit

http://www.worldviz.com/products/vizard/index_b.html

5.1.1) Features

Vizard toolkit is a suite that enables visual editing of 3D environments, enables custom functionality through Python scripting and has good out – of – the – box support for numerous input devices. It also provides a built-in physics (rigid body dynamics) library, accessible through its Python API. It is a combination of an authoring environment with a built – in 3D editor, and a simple development environment for Python development.

5.1.2) Why not?

During experimenting with Vizard we came across some of its aspects that could not support well our course of action:

- The only predefined items of the puzzle world is the surface on which the pieces lie when idle, and the two hand cursors. Everything else (namely the pieces that comprise the puzzle instance) are dynamically created and added to the world, based on the corresponding description files of the puzzle (see §3.4.1 and §3.4.2). In this sense, the editing options of the environment were not really utilized and no productivity improvement gained.
- At the time of experimenting with Vizard, there was no obvious way to couple the rigid body spatial properties (translation, orientation / rotation) with their graphical representation spatial properties counterparts. The composite physical objects would react properly during the world stepping simulation phase, but when moved dynamically in the world, their "physics" position and their rendered position would not be in sync.

- Vizard communication with 3D Mouse devices we tested was immediately successful. On the other hand, the Phantom Omni device we tested proved unreliable in terms of cooperation with the system, although this can't be really attributed to the Vizard environment but to the overall system configuration. For device descriptions, see §2.5.2. This also was one of the technical (that is, not design oriented) reasons, that testing with the Phantom Omni became less of a priority.

5.2) Panda3D Game Engine

<http://www.panda3d.org/>

5.2.1) Features

Panda is a full - blown 3D rendering engine with a strong game bias, which has been used in commercial products. Quoting from the "Introduction to Panda3D" page of the manual [32]:

"Panda3D was developed by Disney for their massively multiplayer online game, Toontown. It was released as free software in 2002. Panda3D is now developed jointly by Disney and Carnegie Mellon University's Entertainment Technology Center "

It is a complete API with many advanced features for rendering, profiling and Python integration.

5.2.2) Why not?

Being (mainly) a game oriented engine/framework, Panda3D has a great deal of underlying functionality that can be restrictive:

- Certain forms of input are by default mapped to game related interaction (for example controller movement is initially mapped to first person camera orientation change / movement – yaw / pitch / roll). This of course can be worked around.
- The overall structure of the world must conform to the internal representation of the engine, namely world items are nodes in a scene graph
- There was no direct way to provide meshes created at run time with appropriate physical properties. The engine is most comfortable with its own data representation (.egg files) or more standard graphics formats (for example .obj files) which do not cater for metadata concerning physics. Also we wanted a method to dynamically create new classes of puzzle pieces, without having to edit an actual 3D mesh to that end.
- Panda3D did not include a dynamics module, integrating with some external collision / physics library would be necessary anyway.

After a brief survey of similar solutions, for example the Ogre3D Open Source 3D Graphics Engine (<http://www.ogre3d.org/>) we concluded that using a readily available 3D engine would be an overkill in terms of functionality on offer, and for the same reason an unnecessary shift of the development focus from a certain small fixed set of requirements to manipulating a large framework and trying to make it comply to our needs.

6) Environment – Software Libraries

Puzzle3D is written in standard C++ with some external dependencies:

- **SDL** – Simple Directmedia Layer (<http://www.libsdl.org/>) is used for easy window creation and event management.

Alternatives:

- SFML - Simple Fast Multimedia Library (<http://www.sfml-dev.org/>)
- GLFW – GL Framework (<http://www.glfw.org/>)
- **OpenGL** (<http://www.opengl.org/>) is the API used for rendering along with the Glee (GL Easy Extension) library (<http://elf-stone.com/glee.php>) for easier manipulation of some OpenGL extensions required for the rendering of shadows. In Microsoft Windows environments, the last supported OpenGL version is 1.1 which is rather old, but more advanced and up-to-date functionality can be achieved through said extensions.

SDL is coupled easily with OpenGL, is mature, stable and used in a large number of projects. It is also “transparent” enough to allow integration with other libraries.

- From the Microsoft **DirectX SDK** (<http://msdn.microsoft.com/en-us/directx>) we used the DirectInput component to communicate directly with the 3D mice we use. DirectInput provides a generic method to obtain data directly from a device, bypassing driver support and other manufacturer details (essentially it conveys data concerning axes and switches)

The implementation of DirectInput cooperation was based on samples from 3dconnexion. <http://www.3dconnexion.com/forum/viewtopic.php?t=1610>, <ftp://ftp-us.3dconnexion.com/>

- **Bullet Physics Library** (<http://bulletphysics.org/wordpress/>)

Bullet is the library used to implement the “physical” behavior of the puzzle pieces. It provides with an easy means to set up a dynamics world and implements the simulation steps required (collisions, dynamics, force response) to update the position of the managed items, which the rendering module can then use to visually represent the positioning of the items in the puzzle world. The simulation flow occurs in a discrete step fashion for each frame tick (at 60 hz), and the updated item state is then used for rendering

Bullet is open source and has been used in major movie productions such as 2012, Sherlock Holmes and various animated films (<http://bulletphysics.org/wordpress/?p=241>)

Alternatives:

- **Newton Game Dynamics** (<http://newtondynamics.com/forum/newton.php>)
- **NVIDIA PhysX** (<http://developer.nvidia.com/technologies/physx>)
- **irrKlang** sound library (<http://www.ambiera.com/irrklang/>) greatly simplifies audio files loading and playback with 3D spatial source and listener properties. It is a commercial library, free for non-commercial use, used (among others) in the well-known game World of Goo.

- For a quick solution on obtaining some simple sound effects for audio feedback we used Tomas Pettersson's (Dr.Petter) **sfxr** application (http://drpetter.se/project_sfxr.html)

7) Software Engineering & Classes

This section provides a brief description of the main conceptual building blocks of the application, and how they are related to each other

class Object

This is the base class of all the interactive items in the puzzle, Cursors and Pieces. It provides the means to change the overall diffuse color of an object (such as setting the color of a Piece when grabbed by a cursor). Each object instance also contains a `btRigidBody` instance that represents the object in the physics simulation, and allows for piece – to – piece and cursor – to – piece interactions.

class Puzzle3D

Implements the application, supervises the Puzzle3D space, organizes input handling and implements logic for loading the puzzle and (partially) constructing the pieces that comprise the puzzle instance. It also manages initialization and disposing of all graphics and audio assets.

class Piece

Instantiates the pieces of the puzzle, is responsible for constructing the actual representation of each composite piece in the physics simulation world, based on a simple piece description. Piece instances are created only if required from the puzzle description file

Bullet Physics (like all physics APIs) are heavily optimized to work with shapes that are basic geometric primitives (boxes, spheres etc), or can be described as compound shapes built from such primitives. In order to exploit this, we assume that each piece is built from a set of cubes of same size “glued” together to form an complex object. The description of each piece type lies in the Shape Library file (see §3.4.1).

class Cube

The main assumption / restriction of the puzzle world is that all pieces can be thought as being built out of smaller cubes. Class Cube implements the rendering of each cube so that every piece can be rendered as a set of simple cube meshes, and maps the owner piece world translation to cube world translation to produce the integer grid coordinates for each cube

class Cursor

The (two) instances of this class represent the user's virtual hands. Regarding user interaction, the simulation is concerned with when the geometry representing each cursor “interacts” with a piece. The cursors are representing as spheres (as the orientation of an empty hand is indifferent), and although they are physical objects concerning the dynamics world (they can interact with other objects, collide, push etc.) they don't take part in the simulation step, as they are moved freely by the user. In this fashion, they are no special objects concerning the implementation (they are also rigid bodies, as the puzzle pieces are) but are treated differently in each frame iteration.

struct PieceDesc

This data structure represents a simple description of a piece in the form of a series of integer 3D coordinates. The piece is always modeled in model space, that is around the origin (0,0,0), with each triplet (coordinate set) representing the placement of a cube. Coinciding cubes should be avoided in the description of a piece for physics simulation stability and correctness, and are not meaningful in a physical sense anyway. Instances of PieceDesc are used to construct concrete piece instances.

Dynamics module

The functions provided Dynamics module provide the means to interact with the dynamics world and implement world creation and shutdown as well as adding items, constraints and so on

8) File Structure

8.1) Shape Library

This text file contains the descriptions of all available pieces. Its simple format allows for more arbitrary shapes to be added without any external modeling. The file is loaded upon application initialization and all respective instances of struct PieceDesc objects are instantiated, but no concrete piece is created. Every piece class also has a string name defined in the file, while an id is automatically assigned to each on load time. The id provided is simply a serial number of the piece classes, so changing the Piece Library after recording a puzzle solution will result to not correctly registering a “solved” state.

Shape library format sample:

```
PIECE Piece1 3
0 0 0
1 0 0
2 0 0
PIECE Piece6 5
0 0 0
0 1 0
0 -1 0
1 0 0
-1 0 0
```

ENDLIBRARY

8.2) Puzzle Description File

The name of the puzzle description file is passed as a parameter in the Puzzle3D configuration file. A recorded puzzle description file contains a specific puzzle instance as a series of the names of the pieces that it built out of, as well as the description of its “solved” state. A solved state is stored as a series of GRID_SLOTS, each with the piece integer id in each slot, followed by its coordinates in the normalized

solution bounding box. For the procedure of recording a puzzle to obtain the said description see 11 “Puzzle Recording and Evaluation”.

Puzzle description format sample:

```
PIECE Piece1
PIECE Piece2
PIECE Piece3

BBOX_SIZ 3

GRID_SLOT 2 2 2 1
GRID_SLOT 2 1 2 1
GRID_SLOT 2 0 2 1
GRID_SLOT 0 1 1 2
GRID_SLOT 0 1 0 2
GRID_SLOT 1 1 0 2
GRID_SLOT 1 0 0 3
GRID_SLOT 0 0 0 3
GRID_SLOT 0 0 1 3
GRID_SLOT 2 0 0 3
GRID_SLOT 2 0 1 3
GRID_SLOT 2 1 1 3
GRID_SLOT 2 2 1 3

ENDPUZZLE
```

8.3) XML Configuration File

The inclusion of an .xml configuration file arose from the need to easily tweak some parameters of the environment, as the usage of a Graphical User Interface (GUI) was considered beyond the scope of the current implementation. However, besides allowing us to fine – tune some of the interface parameters – like controller sensitivity – in order to provide for a better user experience, it became an integral part of the the puzzle application usage process : in the configuration file we define the name of the puzzle file to be loaded by the application and a switch that indicates whether we are in puzzle recording mode (for a description of the procedure see §3.7: “Puzzle Recording and Evaluation”).

Some other parameters defined in the configuration are:

- cubeCollisionScaling : the scale factor for the rigid cube objects as represented in the dynamics simulation module (see §4.1: Simulation vs. user friendliness)
- cubeFriction : the friction that cube objects display against each other (this is different to the friction that cubes display against the ground)
- rotateStep : the step at which rotations are performed (applies to all axis)

Configuration file format sample:

```
<?xml version="1.0" ?>
```

```
<config>
<mouse3DMoveSensitivity value="0.00005"/>
<mouse3DRotateSensitivity value="0.005"/>
<mouseMoveSensitivity value="0.001"/>
<mouseRotateSensitivity value="0.1"/>
<cubeCollisionScaling value="0.96"/>
<cubeFriction value="10.0"/>
<rotateStep value="45.0"/>
<puzzleFile name="Puzzle1.txt"/>
<enableRecording value="true"/>
</config>
```

9) Rendering

In this section the main aspects of the rendering module of Puzzle3D are described

9.1) Rendering Components

The rendering module of Puzzle3D is responsible for displaying all the puzzle building blocks, as well as the space where the interaction takes place and the user means of interaction (the representation of his hands).

9.1.1) Cubes and Pieces

Each piece is formed by a set of cubes. At initialization time, a single instance of the vertex data describing the cube geometry is allocated and initialized, in the form of client-side OpenGL vertex, normal and texture coordinate buffers.

```
glVertexPointer();
```

```
glNormalPointer();
```

```
glTexCoordPointer();
```

As a technical note, this method of geometry data storage is considered deprecated as of OpenGL v3, as client (application) side vertex buffer storage is abandoned in favor of server (driver) side storage. It is still supported in OpenGL 2.0 and earlier execution contexts, where server side storage is accessible through extensions as an optimization mechanism. However for more recent OpenGL contexts, there is no support, and usage of Vertex Buffer Objects (VBO) [12] is mandatory.

Using the older functionality does not impose any penalty on visual quality, and as there would not be any visible performance gain by using VBOs, the transition was left as an optional future upgrade.

The only visual information intrinsic to each cube is the texture index. The application of a “wooden” appearance to the puzzle pieces was mainly an aesthetic choice (inspired by the natural appearance of

similar real – world puzzles – see fig. 10). However, presenting explicit visual clues about the cube topology of each piece, (by means of interchanging light-dark wood texture) proved to be a valuable aid, as it makes relative position and alignment assessment easier for the user.

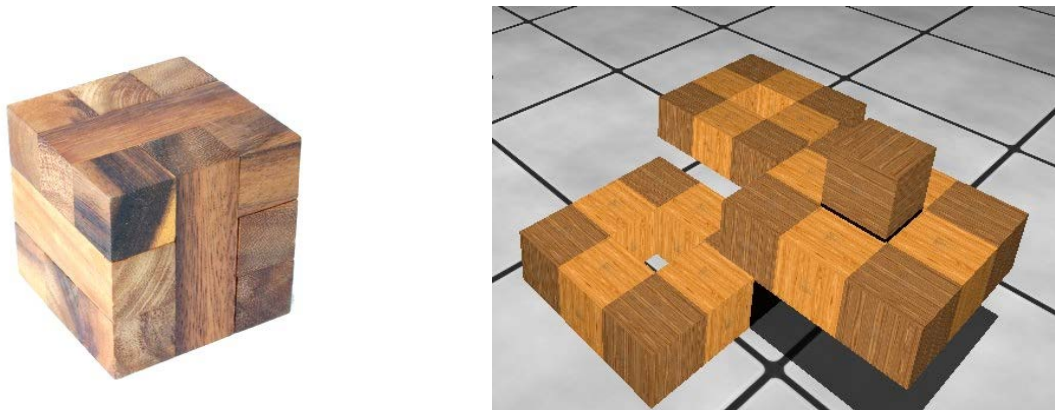


Figure 12: A real-world cube lock puzzle and a rendering of a set of pieces in Puzzle3D

The assignment of texture is done automatically based on the integer cube coordinates in model space using the formula:

```
texture = (x + y + z) MODULO 2 ? GetLightTexture() : GetDarkTexture();
```

in C – like notation, or more descriptively:

```
if (x + y + z) MODULO 2 EQUALS 1
    Use Light Texture
else
    Use Dark Texture
```

where x, y, z are the integer model space coordinates of the current cube.

In order to display each piece, at its correct world position, the world transformation is retrieved from the physics simulator. Conveniently, the Bullet library introduces a function to obtain this transformation in the form of an OpenGL conformant 4x4 matrix

```
void btTransform::getOpenGLMatrix(...);
```

The current piece world transformation is then stored in the OpenGL matrix stack, and for each cube, the local (model space) transformation is applied, the cube geometry is rendered and the world matrix is restored to its previous state to ensure that each cube will be positioned relative to the piece world position.

```
For each piece p
    apply the world transformation matrix WM
    store WM
    for each cube in p
        multiply WM with cube model space transformation CM
```



```
render cube
restore WM
```

9.1.2) Cursors

The cursors were initially conceived as hand – shaped meshes. The modeling would not have to be realistic, as long as the shape would convey the idea of the user's hand int the puzzle space. However, the original implementation as mere spheres with different colors (considered only temporary at the time), allowed us to make some interesting observations, discussed in §4.3.

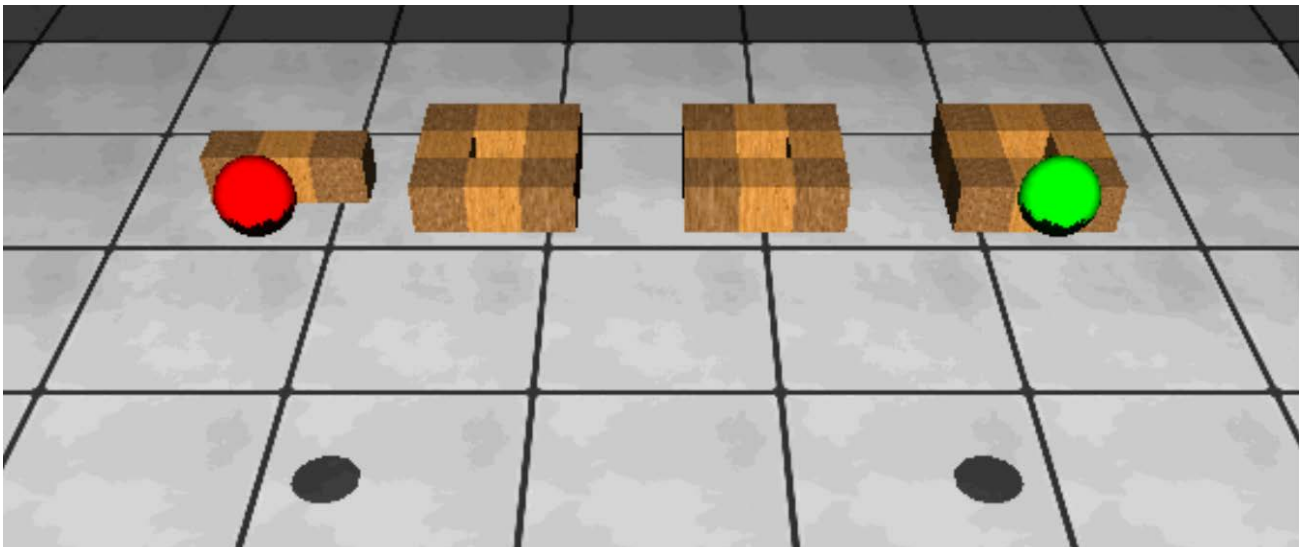
The two cursors are rendered as procedurally generated spheres using the OpenGL utility library (glu) function

```
void gluSphere(GLUQuardic* obj, GLdouble radius, GLint slices, GLint stacks);
```

where the parameters radius, slices and stacks control the geometry generation values of the shpere. If the cursor is carrying a piece then rendering of the sphere is disabled to reduce visual cluttering, and the piece at hand is rendered with the cursor's ambient color to enable easy identification of which cursor is manipulating which piece.

9.1.3) Ground

The ground is rendered as a tiled pattern of smaller textured flat squares. The texture (diffuse map) representing a square pattern was found to make easier the perception of perspective, world scale, and the relative positions of the pieces, as opposed to a seamless texture image.



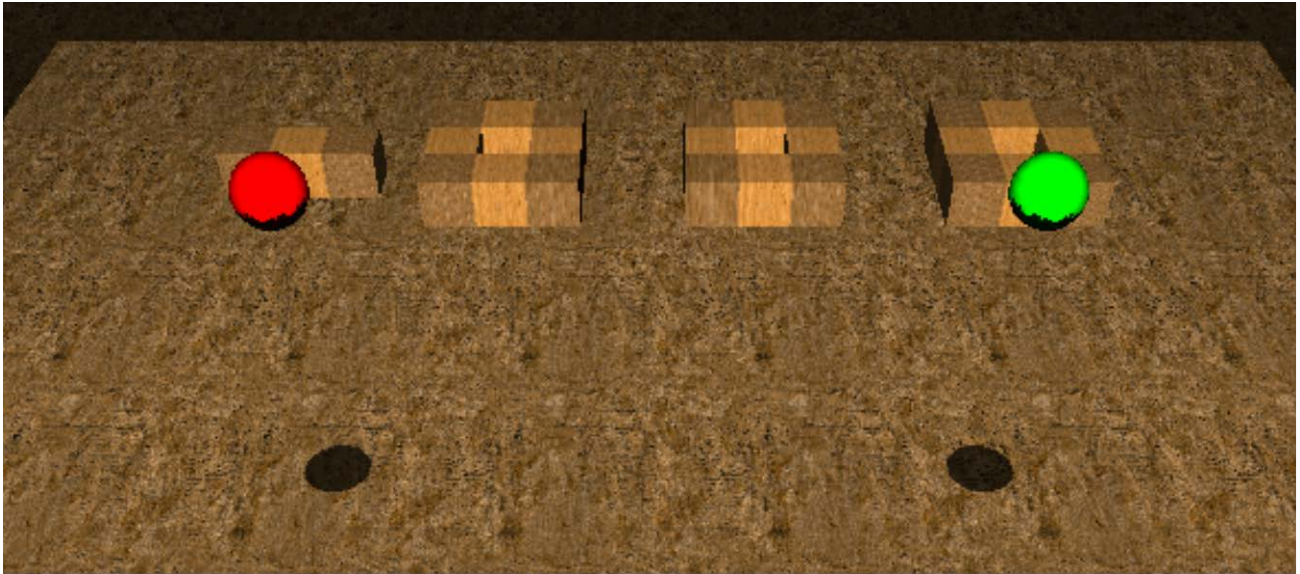


Figure 13: Tiled vs. Seamless texture map on the ground

The ground is not rendered as a single large textured square in order to compensate for the limitations of the light calculation method OpenGL implements by default (Blinn – Phong shading model). This issue can be overcome by the use of a per-pixel lighting method through the OpenGL programmable pipeline, but this was considered beyond the scope of this work.

9.1.4) Camera

At early stages of development, Puzzle3D had no method of changing the user's vantage point. The concept was that the user's view of the world is that of "sitting" in front of a "table" where all the puzzle pieces lie. This notion of the puzzle world however, completely omitted the fact that in the real world, the user has several options for viewing the puzzle from different angles:

- She/he holds the constructed puzzle in one hand and changes the puzzle view by rotating his/her hand
- She/he constructs the puzzle on a surface, but is able to rotate/drag it around without lifting it
- She/he changes her/his own position relative to the puzzle space

Due mainly to reasons concerning simulation stability when binding together a set of pieces, we provided for the last solution. The user can change between piece and camera manipulation mode anytime during interaction. Camera is not free in the sense of a virtual walk – thru mode, but always pointed at the center of the puzzle space (which coincides with the origin (0,0,0)), and distance from the origin also remains constant at all times. The controller can actually be thought of as manipulating the azimuth and altitude of the camera position.

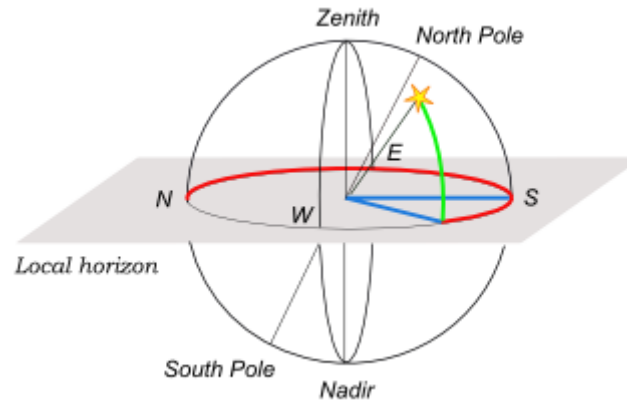


Figure 14 : Horizontal Coordinates. **Azimuth**, from the North point (red) -also from the South point toward the West (blue). **Altitude**, green (source [http://en.wikipedia.org/wiki/Altitude_\(astronomy\)](http://en.wikipedia.org/wiki/Altitude_(astronomy)))

9.2) Shadows

Display of shadows was considered essential for intuitive estimation of the relative positioning of pieces and cursors in the puzzle space. It also adds to the immersion factor considering we strive for a credible 3D workspace.

9.2.1) Method

Single – directional perspective shadow mapping with multitexturing.

This is a multi – pass algorithm that requires the following steps:

1. We render the scene from the light's point of view, with depth – testing enabled. This yields a texture map with a float depth value for each pixel. We are actually interested only in the depth values, so color rendering during this phase is disabled.
2. We render the scene from the user's point of view with dim or no light.
3. We render the scene again from the same point of view with the desired lighting enabled. For each pixel its light view coordinates are calculated. Then, a comparison of the depth computed at step 1 to the current light – pixel distance is performed. If the pixel's distance is greater than the depth value stored in the shadow (texture) buffer, then the pixel is obscured by an object closer to the light, and the frame buffer is not updated to the lit value, leaving the dimly – lit pixel of step 2 on display.

For a more extensive explanation with OpenGL fixed – functionality sample code, see [14], [15]

9.2.2) Limitations and Issues

The implementation of step 1 without any OpenGL extensions, limits us to rendering said texture in the display buffer, which has two main disadvantages:

- It involves more operations on the frame buffer in the course of each frame and

- The resolution of the produced image is limited by the smaller value of the rendering window dimensions. (actually, in order to follow the compatibility convention that texture dimensions should optimally be powers of 2, we are limited to the maximum power of 2 that is smaller or equal to the smaller window dimension)

By using OpenGL extensions, we can overcome both these issues at once, by rendering the “shadow” texture to an off - screen video buffer of any dimensions. Of course, this does not eliminate the need to render the scene, but this action does not involve frame buffer updates. The image generated is readily available as a depth texture.

This method is possible through functionality exposed by the OpenGL Framebuffer Object extension function family [13]

Due to float calculation limitations, shadow outlines may appear jagged primarily in instances where the rendered polygon plane is nearly parallel to the light direction. This can lead to unrealistic – looking shadows. Generally these issues are resolved by using a very large shadow map, performing some type of anti – aliasing method (see [17]), and carefully placing the light source where possible (for example in video games where light – staging is also artistically meaningful)

10) Physics Simulation

In order to simulate the behavior of the puzzle pieces in a realistic manner, we used the open – source Bullet Physics Library. The initial purpose was to give a full “natural” representation of the behavior of the pieces as if they were real objects interacting with each other, with the “ground” plane as well as the cursors. The physics library allows the user (the application developer) to define a large number of parameters and attributes that dictate the physical object movements and interactions. However, for usability reasons we decided to omit some of the naturally expected behavior by disabling certain simulation computations. The particular reasons and the results are discussed in detail in §4.1.

In the following paragraphs, the facilities of the physics library that are used and the procedure to create the puzzle world are presented.

- In order for any entity in the physics simulation world to be able to interact naturally with other entities, it should be described as an object of type rigid body.
- Each rigid body is defined by means of a collision object that describes its geometry plus weight, friction, inertia and so on.

The collision object can be an arbitrary shape formed from a vertex cloud, or a triangle set, or more often than not, built up from a set of simple standard objects like boxes, spheres, cylinders etc. The reason that every physics library includes such a predefined set of simple object types is that there are highly optimized algorithms available to process collisions and interactions among these types.

In our case the pieces are composite shapes built from cubes. This provides us with a simple means to define new piece types (see §3.4.1), as well as makes the definition of the collision shape of each piece very straightforward in our implementation.

- Since it is possible to use the same collision shape in multiple rigid object descriptions, we create the respective collision shape for each piece upon first request to create the piece as a rigid object, use it to

describe the rigid object, and leave it available for the whole lifetime of the application for further requests as a simple form of caching. The pieces are created as rigid objects and added to the physics world in the sequence they are read from the puzzle description file (see §3.4.2).

- We define an immovable horizontal plane to act as the “ground” of the puzzle area, to coincide with the rendered “ground” at $y = 0$ coordinate. As far as the physics simulation is concerned though, the collision plane is infinite. In addition, we add four more vertical planes at reasonable distances from the puzzle space (which is defined around the origin), in order to prevent of pieces being translated to distances that is impractical to manage due to explosive dynamics reaction of the simulation. (see §3.6.1). The total of the five planes form a well – like space of square cross – section.
- Finally the cursors are also defined a rigid body shapes, with the additional parameter of being kinematic objects, that is objects that their behavior is not dictated by the rules of the simulation but rather they display a one – way relationship with other objects as their state is determined by factors outside the simulation (for example user input).

This allows us to create cursors the response of which to the user's input is always straightforward (for example they cannot be encumbered by the presence of other rigid bodies), but also introduces some issues of accuracy and predictability to the system, inherent to the simulation algorithms.

- In order to detect cursor – to – piece collision (when a cursor touches an object, so it can be picked up) we use the built – in collision facilities of the physics engine. When a cursor is detected to “touch” a piece, then this piece is registered as available to pick up, and pressing the respective controller button causes the cursor to be “replaced” by said piece
- Picking is implemented as creating a constraint between the cursor rigid body and the piece rigid body the cursor collides with. Symmetrically, pressing the “pick / select” button again when a piece is at hand, leads to releasing the piece, by “breaking” the constraint - that is removing from the physics world. During the time of carrying a piece, the collision of the carrying cursor and all pieces is disabled.

10.1) Limitations and Issues

The physics simulation relies on a step – based advance of the world. Naturally, all pieces that are simulation – driven are expected to behave in a realistic manner within the specifications of the simulation. The use of kinematic objects to represent the cursors and detect their interaction with other world objects introduces several issues which we discuss here.

At each simulation step, the physics engine calculates the expected position of each rigid body in the world, based on its physical attributes as described upon creation of the body. These attributes include geometry, weight, friction, inertia, world gravity and so on. The simulation step frequency is considered adequate at 60 Hz, or, one simulation step at each frame for a 60 frame – per – second application. This frequency ensures that the simulation caters for situations like objects penetrating each other, and ensures that rigid body motion that results from force interactions inside the world is smooth and predictable.

The issue arising from the use of kinematic objects as cursors is that the motion of cursors does not adhere to the simulation of forces that appear in the world through object interactions. Instead, the position of each cursor is enforced into the world, directly through the input values of each controller. This results in the cursors taking “leaps” into the world – at least from the physics engine point of view – as their position is not a result of step calculation. This in turn, forces the simulation to calculate the new state of the world,

and the forces produced based on a state that is not a direct result of the previous world state, as the rigid bodies of the cursors have arbitrarily changed their position in the world during the input handling state of each frame.

As a result, during the step calculation, a cursor may appear to penetrate a rigid body although this state could not be “foreseen” at the previous step. The physics engine will try to compensate this by applying a repulsion force between the objects, and as the cursor's position in the world is solely determined by input state not taking forces into account, the free rigid body will move explosively away from the cursor.

The behavior described above is probably the most easily observable instance of the issues arising from the more general fact that kinematic cursors can in fact be “anywhere” in the physical world, even inside rigid geometry. For example, similar behavior would arise when trying to “press” a free object against the floor. The cursor would at some time appear “inside” the free object, and the engine would apply a suitable force that would finally cause the object to “blast off”.

Allowing the cursors to move at a speed that was found acceptable for the physics simulation world stepping, and at the same time not too slow to be impractical for the user, provided an adequate solution. As a potential future improvement we would consider using force – based cursor motion (the cursors would also be free rigid objects and motion would be achieved by applying external forces upon them) instead of enforcing the world position, coupled with a force feedback enabled controller. In the absence of force feedback, we chose not to impose any restrictions to the cursor motion (like other objects blocking the cursor) as the “difficulty” to move would only be presented visually, and as such feel too artificial.

11) Puzzle Recording and Evaluation

The goal of the user interaction with our 3D world is to build a construct (the puzzle shape) out of the provided pieces (the puzzle building blocks). As such, there must be a specific predefined shape against which the user's attempts are compared, so that the system can provide the user with feedback about whether she/he has succeeded or not.

Creating a new puzzle is a multi – step procedure:

(a) If the intended puzzle can be created using the pieces provided in the Piece Library file (explained in the file formats section §3.4.1 – Shape Library) then this step is complete, otherwise the puzzle editor must provide the description of the new required shapes in the file.

(b) The editor must then enter the application in a “recording” mode, and specify the puzzle file she/he is working with, which for the moment must contain only the names of the pieces the intended puzzle is comprised of. These options are provided by the configuration file of the application.

There she/he can build up the intended puzzle using direct manipulation like the “player”, and then press “r” (for record) to store the puzzle state.

(c) The player can now enter the application by specifying the puzzle file she/he intends to try to solve. The system will inform her/him when the puzzle has been successfully solved.

The storage of the “correct” puzzle state as well as the process of testing if the user has succeeded was one of the most interesting challenges of the project:

- The user should be able to solve the puzzle in any point of the 3D space. This contrasts the more game-like approach of a “solution” area wherein the user constructs the puzzle in order for his solution to be evaluated (this approach was also suggested by the problem description of the contest). Our intent was to give as few restrictions as possible to the space the user has at his disposal, and the notion of an artificial “solution area” was alienating to the sense of natural space we tried to convey.
- There should not be any assumptions about the orientation of the puzzle. If the correct relative positioning of the pieces is achieved, then the system should report the puzzle as solved, even if it is built, for example, upside-down, or sideways compared to the stored “correct” state. This we believe adheres more to the users' notion of “solving” the puzzle, albeit with the extra limitation that the construct should be structurally sound, as gravity still applies – we do not take into account a solution that could fall apart when resting on a surface.

The methods we considered for storage and solution evaluation are :

11.1) Relative transformations

The description of each piece type contains the information necessary to build a piece in “model space”, meaning each piece is described by the integer coordinates of its building cubes around (0,0,0). The pieces are then placed into their original positions at a setup phase, and their subsequent world positions are dictated by input and simulation. The positioning of each piece in the world is represented by translations and rotations.

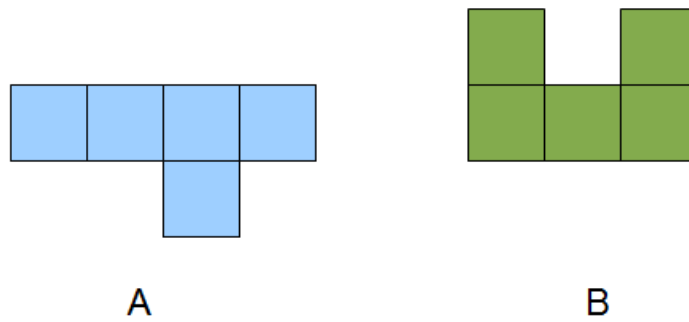
Let M_A be a 4x4 transformation matrix that describes the position of a piece A in the 3D world, and M_B similarly for a piece B.

The relative transformation from M_A to M_B would be a 4x4 matrix X with the property:

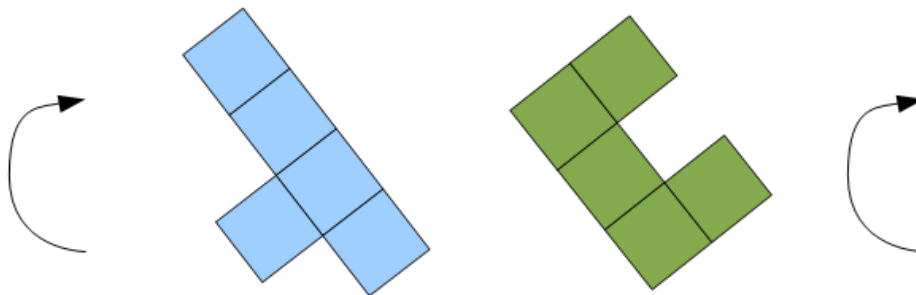
$$AX = B \rightarrow X = A^{-1}B$$

So, given two pieces at given positions, (that is given the transformation matrices that place them in these positions) we can produce the position of each relative to the other. Storing these relative positions for every pair of pieces comprising the puzzle, we should have an adequate description of the solved state, which moreover would be puzzle orientation agnostic, as the orientation of the pieces is only tested against each other.

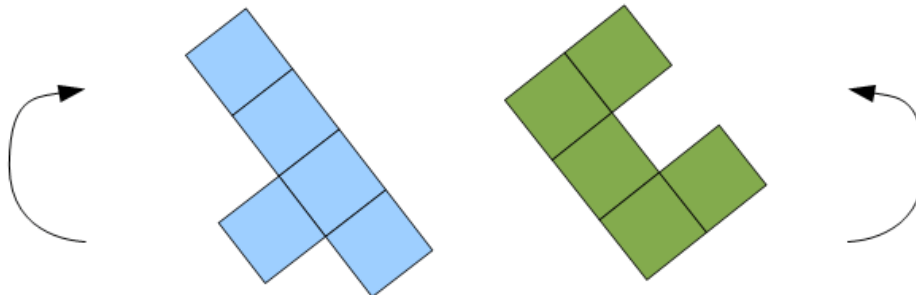
This solution however could not take into account the fact that some orientations that would be different from a numerical perspective (different matrix representations), would appear equivalent due to piece symmetry. (The following example is displayed in 2 dimensions for simplicity)



a) Initial piece positions (model space cube coordinates)



b) Both pieces rotated clockwise by 45 degrees



c) Same result achieved with piece A rotated clockwise by 45 degrees and piece B rotated counter-clockwise by 315 degrees

Figure 15: An example of relative positioning equivalence

This implies either that the user would be required to place the pieces in a strictly predefined fashion, or the system should evaluate against all alternatives.

The first case would be a “system” imposed limitation to a “real-world” simulation (and thus unacceptable), as the user does not naturally differentiate between the two states of the piece in Figure 12.

The latter case would be impractical for a real-time application, as the number of alternatives to a solution can be very large. Consider a puzzle containing n pieces that present such a symmetry that there are

k equivalent transformations for each. Then the alternative puzzle configuration descriptions are k^n assuming that there are no other symmetrical pieces.

11.2) Contact Points (cube faces)

Another possible solution we considered is the “contact points” approach. In this, every cube face that matters to the solution evaluation (that is the cube faces that form the hull of each piece) would be marked with an id, and its orientation in the world space represented with its normal vector.

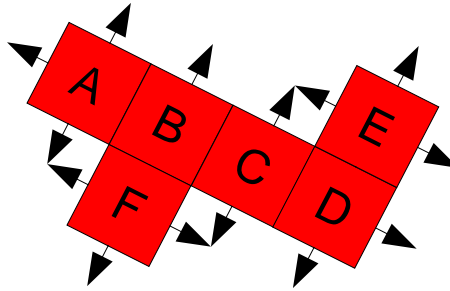
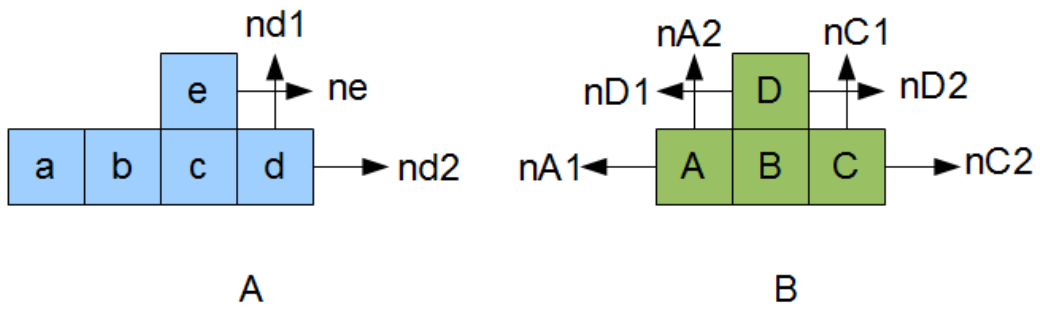


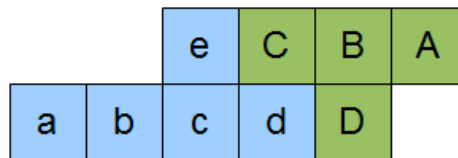
Figure 16: An 2D representation of a composite shape with hull normals

To perform contact point matching, for any given orientation of the pieces in world space, we could match the face normals taking into account a predefined tolerance value, and compare the matching sets with our prerecorded solution. The tolerance value (which applies in the same fashion for the Relative Positioning solution) is needed to amortize the need for rigidly precise piece placement on behalf of the user, and to compensate for the innate floating point round off error of the system. If the origins of two normal vectors coincide within this tolerance limit, and they have opposite orientation, then the owner cube faces are in contact.

The problem arising in this solution has a similar cause to Relative Positioning. For any given piece, it would be necessary to have a method to detect axis or plane symmetries and to mark the cube faces accordingly in order to provide for alternative piece placements that “construct” the (visibly) same solution.



a) The pieces appear as described in model space.
Only selected normals are shown to avoid image cluttering



b) Piece B rotated 180 degrees around the negative z axis (pointing from the reader towards the page)
Matching normal pairs are $ne - nC2$, $nd1 - nC1$, $nd2 - nD2$



c) Piece B rotated 180 degrees around the x axis.
Matching normal pairs are $ne - nA1$, $nd1 - nA2$, $nd2 - nD1$

Figure 17: An example of contact point matching equivalence

Although this approach would result in a fairly generic system that could support a wider variety of shapes and contact scenarios, it was considered to be beyond the scope of the present work, mainly due to complexity issues.

11.3) Grid matching

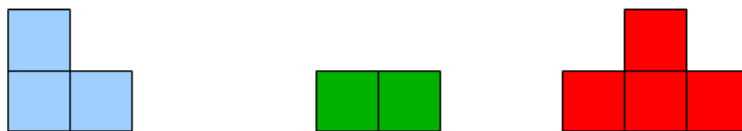
The solution we implemented has the advantage of overcoming the problem of symmetries in a simplistic but effective manner, by removing the actual cube topology that forms each piece from the solution description. Each cube of any piece “class” (type) is marked with an integer id that represents this class.

For each simulation step a set of integer 3D coordinates is formed from the (floating-point) world position of the cube as produced by the world transformation of each piece. Based on the integer cube coordinate set, an axis-aligned integer bounding box is formed that includes all available pieces.

If the current bounding box is not of the same size as the one recorded to the piece solution file, solution matching is stopped (as the current state can't be a possible solution). If the bounding box sizes match, then a grid comparison is performed between the grid representing the solution (which is constructed at initialization time from the puzzle description file) and the grid produced by the current world state.

The (x,y,z) grid coordinates of each grid “slot” are an integer mapping of each cube's world position, and its value is the id of the piece class the respective cube belongs to. In order not to restrict the user to a solution area, the grids (and consequently the represented cube coordinates) are normalized to the space $xyz_{min} = (0,0,0) - xyz_{max} = (bbox_size, bbox_size, bbox_size)$ where $bbox_size = \text{Max}(bbox_xsize, bbox_ysize, bbox_zsize)$. Obviously, the bounding box is not tight-fit but rather the minimal bounding cube.

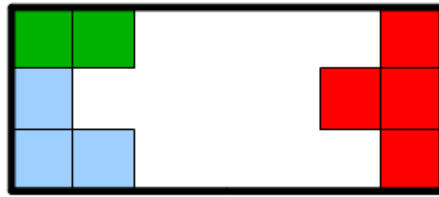
If every id in the respective (x,y,z) grid coordinates is matching then we have reached a “solved” state. In order to be able to detect correct piece placements but with different orientations, we pre-compute all possible permutations of the recorded solution at initialization time, and compare to all of them whenever necessary.



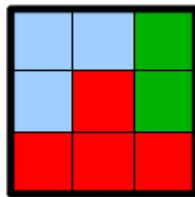
a) Available piece set



b) Recorded solution state (left), and one of the alternatives generated during puzzle initialization (recorded state rotated around the negative z axis by 90 degrees)
Bold line denotes the calculated Bounding Box



c) A puzzle state during interaction. Bounding Box does not match the recorded size so the current state is early-discarded (no individual cube checking takes place)



d) A correct state

Figure 18: Puzzle solution evaluation with grid matching

A major disadvantage of this solution is that in its present form can effectively match puzzle solution states only if the puzzle layout in the working surface is axis aligned. We compensate for this by allowing a fixed angular step to the piece rotations during manipulation, as well as removing local inertia from the piece dynamics attributes, in order to force a more predictable (but less “natural”) piece behavior. (see also §12, “Simulation vs. user friendliness”) A solution that could apply to any angle is (potentially) left for future work.

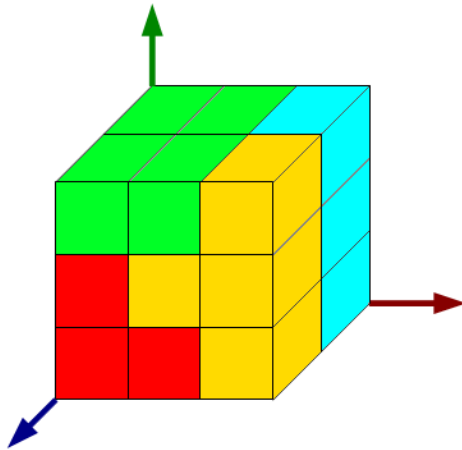
11.3.1) Determining the alternative solutions

Since cube coordinates in the solution description are integers, calculating all possible variants of a given solution is a deterministic procedure – that is it doesn't suffer from float arithmetic errors or similar inaccuracies. As such, generating and storing all solutions in the puzzle description file would be a waste of space. Instead, we store only one solution – the integer coordinates determined by the puzzle layout at the time “record” is executed – and generate all alternatives at program initialization time.

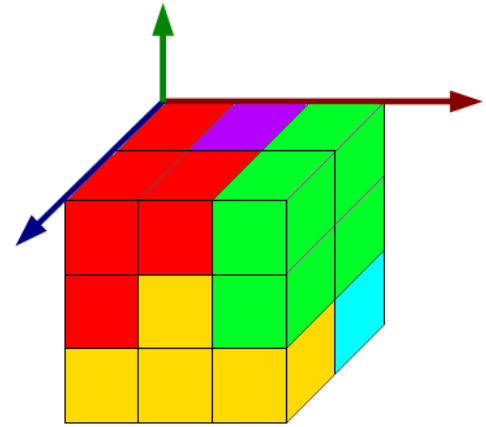
Creating the alternatives, involves rotating the coordinates of the initial solution in order to take snapshots of every rotation around the cardinal axes the puzzle can be constructed with. This of course may also take into account puzzle states that would not be structurally possible, given the physics simulation (states that the pieces could fall off), but determining in advance the structurally sound states, would not only be too complicated, but also the structural “soundness” itself is not a generic term, as it depends on the physics simulation parameters, which may be altered (for example if we disable gravity and inertia, all states are stable).

In any case, this particular approach was chosen because it provides a fixed small number of alternatives to take into account. By fixing all solutions to be axis – aligned, given a single solution state, we

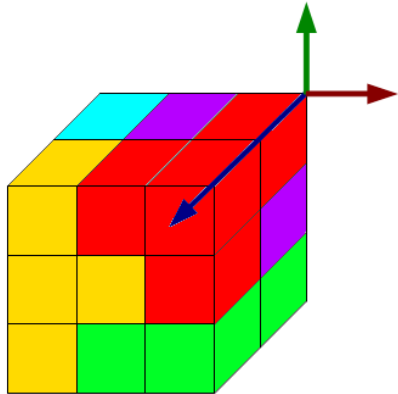
have to rotate the initial solution coordinates around all 6 directions (negative x, positive x, negative y and so on) by 0, 90, 180 and 270 degrees, yielding a total of $6 \times 4 = 24$ alternatives. Any given puzzle state is matched against all of these configurations, but only if the bounding box size implies a potential solution state, as described in 11.3.



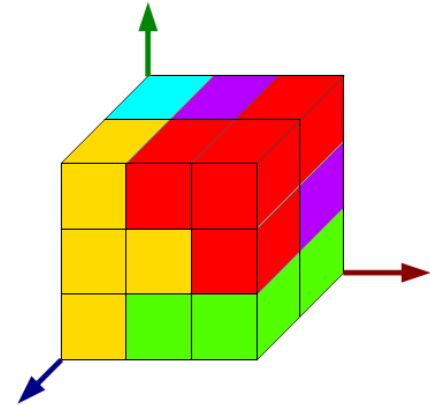
a) The initial solution state as recorded



b) State (a) rotated 90 degrees around the negative z axis



c) State (b) rotated 90 degrees around the negative z axis



d) State (c) as normalized bounding box coordinates

Figure 19: Constructing alternative puzzle solutions - Red arrow denotes positive x axis, Green arrow denotes positive y axis and Blue arrow denotes positive z axis

In figure 16 the process is displayed : alternative solutions are created by 90 degree rotations around the cardinal axes. As the coordinates produced are bound to contain negative values, we normalize each coordinate set to lie inside a bounding cube inside the intersection of positive half spaces with its origin on (0,0,0) thus yielding positive only values.

Part 4 – Lessons Learned: Conclusions and Discussion

12) Simulation vs. user friendliness (or Is there too much physical representation?)

The Bullet Physics Simulation (and indeed any modern dynamics simulation framework) defaults to a behavior that most closely resembles what we would expect from a real-world set of interacting physical objects. When the objects are correctly modeled in terms of their physical attributes (relative sizes and weights, friction, inertia etc.), the simulation does its best to represent a credible representation of their behaviors in the given context. This has presented a tradeoff between the need of a natural appearance that would better convey the sense of interaction with physical objects, and the ease of the user achieving his goal. Given a “precise” physical simulation, several interaction issues arise:

- Lacking a tactile user feedback, the user has no other means of determining a proper fit between pieces – especially concerning proper alignment – besides visuals. In the case of very small differences in orientation (within the range of a few degrees), there must be special (in the sense that they should override the simulation) methods to detect these nearly-correct situations, while being transparent to the user. The reason is that leading the world into a state that would be unacceptable to the correctness of the physical simulation (for example forcibly moving two rigid objects into one another, can cause explosive response which more than unexpected, is unnatural and can be frustrating to the user especially when trying to perform precise operations – see §10.1.

This issue was resolved by forcing all rotations to a 45 degree step. This imposed restriction makes the perception of orientation/alignment much more intuitive and also simplifies the positioning of the pieces into the world and consequently the construction of the puzzle.

- Inertia calculations would cause the pieces to bump on the surface and on each other, and rotate accordingly when hitting an obstacle. Although this adds to the natural appearance of the simulation, it caused the pieces to end up resting on arbitrary angles. It also added another factor of instability when constructing the puzzle, as angular factors are computed based on center of mass, and allowing almost – correctly positioned pieces to remain within the puzzle construct in a stable fashion requires very careful inertia/friction balancing.

In order to maintain a consistent view of the puzzle world, inertia was disabled. This causes the pieces to maintain the orientations defined by the hand cursors when left to fall. The advantage is that the sense of orientation/alignment holds for all pieces, held (in the cursors) and free (on the “floor”) alike. The disadvantage is that it introduces visual peculiarities like pieces not falling flat to a stable state.

- The cubes that comprise the pieces are somewhat smaller (about a factor of 0.96) in their physics simulation representation than their rendered instances. This allows for some “slack” when positioning the pieces, which although not computationally accurate, conveys the correct behavioral “feeling” especially when sliding a piece between others or expecting a rotation within some construct to be feasible.

In all the above cases, we deemed the user experience of greater importance than the correctness and plausibility of the underlying simulation. We believe however that in both cases the advantage in ease of use, world perception and general “feeling” outweighs the drawbacks, a common conception in Human-computer interaction aspects, including video games.

13) Handedness and emergent usage

Using our system with both (symmetrical) input controllers as intended, reveals that the way the user utilizes his (cursor) hands actually evolves towards a mapping of the tasks involved to manipulating the puzzle in real life to the most similar tasks allowed by the system – that is after a certain time allowed to get accustomed to the controls. More precisely, as the controllers provide full symmetry to the actions allowed, common ways to interact with the puzzle world are:

- Using one hand to hold/move/rotate one piece while using the other to operate the camera to shift the puzzle view.
- Using one hand to hold/move/rotate one piece while using the other to hold another piece in a more static fashion in order to provide for stability or resistance to the puzzle structure. This of course also brings up the physics simulation stability/precision issues discussed in §10.1.

It's interesting to remind here that said “one” and “other” hands are completely interchangeable in terms of handedness (our design as discussed in §3.1), at any time during the interaction, so the user's preference of usage is emergent.

It should also be noted that concerning the former observation, the user actually maps the 'typical' means of bi-manual method of constructing objects from parts: one hand holds the construct in progress in order to easily observe it from any angle to determine possible contact points for following parts, while the other usually manipulates a single piece testing possible fittings.

14) The Cursors: Form vs. Function

It has already been mentioned in (3.5.1.b) that displaying the cursors as simple colored spheres was considered only a temporary solution. However, it eventually allowed for some interesting issues to surface:

- The orientation of the cursors is of no importance as long as they are empty – that is not holding a “piece”. So the minimalistic representation as spheres was adequate. Of course, when a piece is held, the sphere is not shown and the orientation is apparent by the piece rotation.
- There is no limitation as to where the cursors can be in the 3D space, so there may be cases in which position of the cursors can be cross-over relative to the users hands from the current view. Moreover, the apparent handedness, though “hand-consistent” from a given point of view, may be switched by a camera change. Fixing the user's view of the world, to enforce handedness was found impractical, as discussed in (4.4 Viewing the world). As a result, depicting the user's hands in an accurate manner, would appear rather alienating (as they would not follow the notion of actually being the user's hands shown in “front” of him) instead of adding to the realism and immersion.
- Due to the previous observation, the colors of the cursors strives not to convey the conventional notion of handedness (left – right) but rather to enable the user to easily “connect” each cursor with a controller. As such the mental model is “my left hand controls the red sphere” rather than “my left hand controls the sphere that appears left on the screen”.

15) Viewing the world

In order to construct the puzzle at hand, the user should have the freedom to view the results of his actions from various points of view. Towards this end we considered two alternatives:

- The user's point of view of the world is fixed. We would try to simulate the experience of holding the “constructed” part of the puzzle in one hand (as a set of pieces) and allow one “free” cursor to manipulate a single piece at a time and search for possible placements on the constructed set. The cursors being functionally equal, the process would be interchangeable in terms of handedness.

This solution though conceptually intuitive proved impractical due to the way the physical simulation allows us to construct constraint – bound sets of objects.

- The user switches between manipulating the cursors and manipulating a camera at any given time. The world view is dynamic, through smooth camera movement and the constructed set of items (the puzzle being built) lies on the “ground”. The can change mode by pressing a button on any controller.

With this method, we achieve the desired result (the user being able to have a dynamic view of the puzzle) without imposing further complexity to the UI – as the user would in either case have to manipulate the movement controls, while simplifying the implementation of puzzle recording and evaluation.

16) “Natural” vs. Familiar

One of the main goals of this work was to experiment with alternative methods of input towards a more natural means of conveying our intentions to the computer system. Towards this end, there is a great number of input devices beyond what we are used to consider “common”, rapidly becoming standards mainly in entertainment, and even more experimental solutions in development and academia. (see 4. Input devices)

However it was interesting to notice, as early tests indicated, that users were much more comfortable to perform any task using familiar controllers such as the 2D desktop mouse, even when confronted with tasks that we felt were more suited to 3D controllers, due to its more intuitive natural mapping. This leads to the assumption that the leap towards using a new device is much greater than the mental shift required to map a familiar device to an uncommon task. In other words, the user seems to find much easier to use well – known tools to solve unfamiliar problems, than to take the time to familiarize himself to a new tool from the ground up in order to perform a well defined task, even if the tool is designed towards this goal and ultimately proves to be more efficient. It also seems that the computer system use context is strongly coupled with the input devices of daily use: using the mouse is “canonical” in terms of interacting with a computer, something that doesn't apply to other devices – yet.

Of course, 20 years ago this unfamiliarity was also the case with the common mouse – at least for PCs, so we can assume here that adoption of new methods of interaction is only a matter of time, provided that is that they prove practical enough.

References

1. Call for 3DUI Contest, IEEE Symposium on 3D User Interfaces 2011
<http://conferences.computer.org/3dui/3dui2011/cfp-contest.html>
2. Yong Wang, Sankar Jayaram, Uma Jayaram. “*Physically Based Modeling in Virtual Assembly*”, The International Journal of Virtual Reality Vol.5, No1 (2001)
3. Sheldon Andrews, Javier Mora, Jochen Lang, Won Sook Lee. “*HapticCast: A Physically-Based 3D Game with Haptic Feedback*”, SITE, University of Ottawa, website:
<http://hapticast.sourceforge.net/>
4. Arkane Studios website <http://www.arkane-studios.com/uk/home.php>
5. Hyeongmook Lee, Mark Billinghurst, Woontack Woo. “*Two-handed tangible interaction techniques for composing augmented blocks*”, Virtual Reality DOI 10.1007/s10055-010-0163-9 SI : Augmented Reality
6. Jun Murayama, Laroussi Bougrila, YanLin Luo, Katsuhiro Akahane, Shoichi Hasegawa, Beat Hirsbrunner and Makoto Sato. “*SPIDAR G&G: A Two-Handed Haptic Interface for Bimanual VR Interaction*”, Proceedings of EuroHaptics 2004, Munich Germany, June 5-7, 2004
7. Renaud Ott, Daniel Thalmann, Frederic Vexo. “*Haptic Feedback in Mixed-Reality Environment*”, The Visual Computer Manuscript
8. André Kunert, Alexander Kulik, Anke Hackauf, bernd Fröhlich. “*A Comparison of Tracking- and Controller-Based Input for Complex Bimanual Interaction in Virtual Environments*” IPT-EGVE Symposium (2007)
9. Ken Hinckley, Mary Czerwinski, Mike Sinclair. “*Interaction and Modelling Techniques for Desktop Two-Handed Input*”, UIST '98. San Francisco, CA
10. Kiyoshi Kiyokawa, Haruo Takemura, Yoshiaki Katayama, Hidehiko Iwasa, Naokazu Yokoya. “*VLEGO: A Simple Two-handed Modeling Environment Based on Toy Blocks*”, [...]
11. Robert Y. Wang, Jovan Popvic. “*Real – Time Hand – Tracking with a Color Glove*”, Jovan Popovic's homepage <http://www.cs.washington.edu/homes/jovan/>
12. The OpenGL Wiki Vertex Buffer Object description, http://www.opengl.org/wiki/Vertex_Buffer_Object, and extension specification http://www.opengl.org/registry/specs/ARB/vertex_buffer_object.txt
13. The OpenGL Wiki Framebuffer Object description, http://www.opengl.org/wiki/Framebuffer_Object, and extension specification http://www.opengl.org/registry/specs/ARB/framebuffer_object.txt
14. Shadow mapping on Wikipedia, http://en.wikipedia.org/wiki/Shadow_mapping

15. Paul's Projects shadow mapping, theory & sample code, <http://www.paulsprojects.net/tutorials/smt/smt.html>
16. Michael Bunnell - NVIDIA, Fabio Pellacini - Pixar Animation Studio, Shadow Map Antialiasing, *"GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics"* Chapter 11, nVIDIA, Addison – Wesley Professional (April 1, 2004)
17. T. Theoharis; G. Papaioannou; N. Platis; N.M. Patrikalakis. *"Graphics and Visualization: Principles & Algorithms"*, A.K. Peters, 2008 ISBN 1568812744, 9781568812748
18. Eric Haines, Tomas Akenine-Moller, *"Real-Time Rendering (2nd Edition)"*, A K Peters/CRC Press; 2 edition (July 2002), Chapter
19. Department of Computer Science and Center for Human Computer Interaction, Virginia Tech *"Building Blocks"*, video http://www.youtube.com/watch?v=5aRJJfR01tM&feature=player_embedded
20. Electrical and Computer Engineering, University of British Columbia A 3D Cubic Puzzle in pCubee, video http://www.youtube.com/watch?v=VUZ6FzkibpQ&feature=player_embedded
21. Hasselt University Look Mother, Virtual Puzzling without Buttons! - Explanation video http://www.youtube.com/watch?v=SlD-wFTBQbs&feature=player_embedded#at=38
22. Reality-based User Interface System blog, includes contest video <http://blog.ruisystem.net/>
23. Dmitri Shuralyov and Wolfgang Stuerzlinger (Team "York Red"), Dept. of Computer Science and Engineering, York University, Toronto, Canada *"A 3D Desktop Puzzle Assembly System"*, www.cse.yorku.ca/~wolfgang/papers/3dpuzzle.pdf, video at <http://www.youtube.com/watch?v=bG1L6YlIKBI>, Wolfgang Stuerzlinger's homepage: <http://www.cse.yorku.ca/~wolfgang/>
24. Hokkaido University, 3D Puzzle Guidance <http://www.youtube.com/watch?v=WsCOi7-bHh8>
25. immersion, creators of the Cubtile <http://www.cubtile.com/>, product page http://www.cubtile.com/index.php?option=com_content&view=section&layout=blog&id=1&Itemid=2&lang=en
26. immersion, 3D Interaction for Puzzle Solving with the Cubtile, a 3D Multitouch Device, video <http://www.youtube.com/watch?v=wXBqWz0LfbC>
27. pCubee web site <http://www.cubee.ca/>
28. Ian Stavness, Billy Lam, Sidney Fels, Department of Electrical and Computer Engineering University of British Columbia, Vancouver, Canada : *"pCubee: A Perspective-Corrected Handheld Cubic Display"* CHI 2010: Pointing and Selecting, source [27]
29. Billy Lam, Ian Stavness, Ryan Barr, and Sidney Fels, Department Electrical and Computer Engineering University of British Columbia Vancouver, BC, Canada *"Interacting with a Personal Cubic 3D Display"*, source [27]

30. Instituto de Informatica, Universidade Federal do Rio Grande do Sul, The Cube of Doom Video I: a Sample User Solving a Puzzle <http://www.youtube.com/watch?v=yvcqp03tlpw>, The Cube of Doom Video II: a Bimanual Perceptual User Experience <http://www.youtube.com/watch?v=xY1cXVOa88M>
31. Peter Brandl, Clifton Forlines, Daniel Wigdor, Michael Haller, Chia Shen *“Combining and Measuring the Benefits of Bimanual Pen and Direct-Touch Interaction on Horizontal Interfaces”*
32. Panda3D Manual : Introduction to Panda3D http://www.panda3d.org/manual/index.php/Introduction_to_Panda3D
33. Peter Brandl, Clifton Forlines, Daniel Wigdor, Michael Haller, Chia Shen, *“Combining and Measuring the benefits of Bimanual Pen and Direct - Touch Interaction on Horizontal Interfaces”*
34. Renaud Ott, Daniel Thalmann, Frédéric Vexo. *“Haptic Feedback in Mixed – Reality Environment”*, The Visual Computer Manuscript
35. Can Ozmen, Selim Balcisoy. *“Bimanual Interactive Tools for Cultural Heritage Researchers”*
36. Patrick Reuter, Guillaume Riviere, Nadine Couture, Stephanie Mahut, Loic Espinasse. *“ArceoTUI – Driving Virtual Reassemblies with Tangible 3D Interaction”*, ACM Journal on Computing and Cultural Heritage, Vol.3 No.2, Article 4, Publication date: June 2010
37. Richard J. Adams, Daniel Klowden, Blake Hannaford. *“Virtual Training for a Manual Assembly Task”*
38. Guillaume Riviere, Nadine Couture, Patrick Reuter. *“The activation of modality in virtual objects assembly”*, J Multimodal User Interfaces (2010) 3:189-196, DOI 10.1007/s12193-010-0038-0