ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Μετατροπή Cube map σε Fisheye

Διπλωματική Εργασία

του

Κωνσταντίνου Σπανάκη

Υπεύθυνος Καθηγητής: Ηλίας Χούστης
Επιβλέπων Καθηγητής: Αθανάσιος Γκαϊτατζής

Περιεχόμενα

Περίληψη

Σκοπός αυτής της διπλωματικής ήταν η ανάπτυξη μιας μεθόδου για την μετατροπή ενός cube map σε fisheye. Η μέθοδος αυτή κυρίως εφαρμόζεται στην μετεωρολογία, στην αστρονομία για την παρατήρηση του ουράνιου θόλου και γενικότερα σε εφαρμογές όπου είναι απαραίτητη η ευρεία όραση για την παρατήρηση ενός τόπου παράλληλα με την εστίαση σε ένα συγκεκριμένο σημείο αυτού του τόπου. Σύνηθες παράδειγμα οι κάμερες που χρησιμοποιούν σε ντοκιμαντέρ.

Πιο συγκεκριμένα, στην ενότητα 1 γίνεται μια γενική παρουσίαση της προβολής fisheye. Ύστερα στην ενότητα 2 γίνεται μια αναλυτική περιγραφή της προβολής fisheye και των διάφορων παραλλαγών του. Στην ενότητα 3 γίνεται μια γενική επεξήγηση του cubemap και εν συνεχεία των διάφορων εφαρμογών του. Η παρουσίαση και γενική επεξήγηση του κώδικα γίνεται στην ενότητα 4, της οποίας έπεται η ενότητα 5 όπου παρουσιάζονται κάποια συμπεράσματα για την εν λόγω διπλωματική.

Λέξεις Κλειδιά: Cube map, Fish eye, Texture Mapping, Non-linear projections

1 Fisheye

Το Fisheye είναι ένα είδος προβολής, όπως η προοπτική και η ορθογραφική προβολή. Σε αντίθεση όμως με αυτές, ανήκει στην κατηγορία των μη γραμμικών προβολών(non-linear projections).

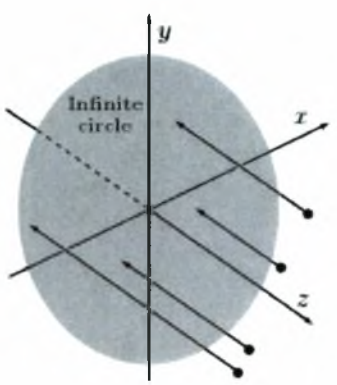
Για την ακρίβεια, μη γραμμικές προβολές ονομάζονται οι προβολές που δεν εκφράζονται βάσει γραμμικών μετασχηματισμών όπως $x^*=ax+cy+m$ και $y^*=bx+dy+n$. Αυτού του είδους οι προβολές χρησιμοποιούνται λόγω των ιδιαίτερων εφέ που έχουν. Επειδή είναι μη γραμμική προβολή, δεν γίνεται να αναπαρασταθεί βάσει πίνακα μετασχηματισμού.

Το όνομα αυτό προέρχεται από τους ομώνυμους φακούς κάμερας που χρησιμοποιούνται από πολλούς φωτογράφους. Η ονομασία αυτή προκύπτει από το ιδιαίτερό τους σχήμα που θυμίζει "μάτι ψαριού". Η βασική ιδέα είναι να πάρουμε το ημισφαίριο, ενός χώρου με άπειρη ακτίνα, τοποθετημένο μπροστά στον παρατηρητή και να το προβάλλουμε σε έναν επίπεδο κύκλο. Επειδή το ημισφαίριο είναι άπειρο και ο κύκλος πεπερασμένος και ίσως μικρός, η προβαλλόμενη εικόνα παραμορφώνεται. Η ομοιόμορφη σμίκρυνση της εικόνας θα κάνει δυσκολότερη την θέαση των λεπτομερειών επειδή θα είναι πολύ μικρές. Μια καλύτερη ιδέα είναι η εφαρμογή μη γραμμικής σμίκρυνση που θα γίνεται εμφανέστερη καθώς μετακινούμαστε από το κέντρο της εικόνας προς τα έξω. Αντικείμενα κοντά στο κέντρο της εικόνας είναι εμφανέστερα στον παρατηρητή και σμικρύνονται λίγο. Η δε σμίκρυνση αυξάνεται για αντικείμενα που βρίσκονται μακριά από το κέντρο. Παρακάτω θα δούμε ότι υπάρχουν παραπάνω από ένα είδη προβολών fisheye.

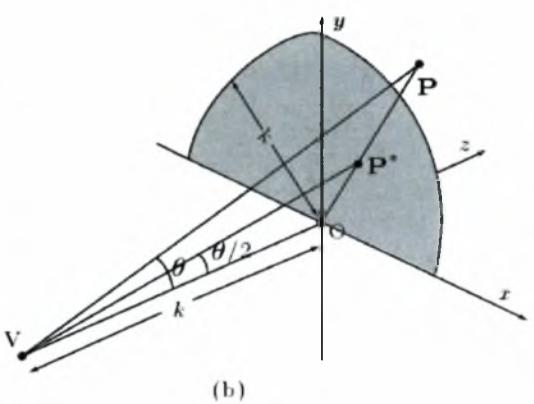
2 Είδη προβολών Fisheye

2.1 Hemispherical Fisheye

Το hemispherical fisheye είναι εύκολο στην κατανόηση αλλά απαιτείται η γνώση τόσο της εφαπτομένης όσο και της αντίστροφης εφαπτομένης του εκάστοτε προβαλλόμενου σημείου. Η όλη διαδικασία γίνεται σε 2 βήματα. Στο πρώτο βήμα όπως φαίνεται στην εικόνα 1, όλα τα σημεία του ημισφαιρίου όπου το z είναι μη αρνητικό προβάλλονται σε έναν άπειρα μεγάλο κύκλο στο επίπεδο xy, με κέντρο το σημείο (0,0). Στο δεύτερο βήμα (εικόνα 2), όλα τα σημεία πάνω σε αυτόν τον κύκλο μετακινούνται προς το κέντρο και καταλήγουν πάνω στο κύκλο ακτίνας k, με κέντρο το σημείο (0,0).



Εικόνα 1



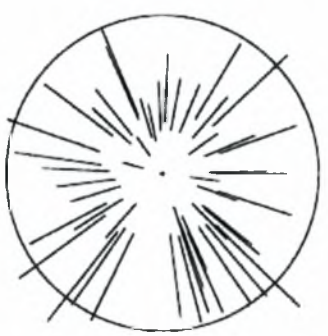
Εικόνα2

Πιο συγκεκριμένα, στο πρώτο βήμα γίνεται παράλληλη προβολή των σημείων στο επίπεδο xy, δηλαδή το εκάστοτε σημείο (x,y,z) προβάλλεται στο σημείο (x,y,0) στον άπειρο κύκλο. Στο δεύτερο βήμα συμπιέζεται ο άπειρος κύκλος σε έναν κύκλο ακτίνας k μη γραμμικώς. Ο χρήστης επιλέγει μια θετική τιμή k και κάθε σημείο στο επίπεδο xy μετακινείται προς το σημείο (0,0) κατά το μισό της γωνίας θ όπως φαίνεται από το σημείο (0,0,-k). Στην εικόνα 2 βλέπουμε το σημείο P στον επίπεδο xy όπου η γωνία μεταξύ της γραμμής VP και του άξονα z είναι η γωνία θ. Το σημείο P μετακινείται πάνω στην γραμμή PO και γίνεται P^* με γωνία θέασης θ/2. Αφού και τα 2 σημεία είναι επί του επιπέδου xy, τότε θεωρούμε ότι έχουμε μετασχηματισμό 2 διαστάσεων. Το

μετασχηματισμένο σημείο P^* ισοδυναμεί με sP όπου s παράγοντας σμίκρυνσης. Βέβαια είναι εμφανές ότι σημεία που βρίσκονται μακριά από το σημείο (0,0) θα υποστούν μεγαλύτερη μεταβολή από τα σημεία που είναι πιο κοντά. Άρα ο παράγοντας s είναι μια μεταβλητή εξαρτημένη από το εκάστοτε σημείο P πράγμα που εξηγεί γιατί δεν είναι γραμμική αυτού του είδους η προβολή. Ο υπολογισμός του s φαίνεται στην εικόνα 2 όπου φαίνεται ότι $\tan\theta = |P|/k \Rightarrow \theta = \arctan(|P|/k)$. Αναλόγως για το μετασχηματισμένο σημείο προκύπτει $\tan(\theta/2) = |P^*|/k$, οπότε ο παράγοντας s ισοδυναμεί με $s = |P^*|/|P| = k^* \tan(\theta/2)/|P| = k^*\tan(\arctan(|P|/k)/2)/|P|$.

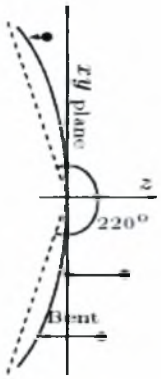
Αν παρατηρήσει κανείς στην εικόνα 2 προσεκτικά, θα δεί ότι τα μακρύτερα από το (0,0) σημεία έχουν γωνία θ ίση περίπου με 90^o. Άρα οι προβολές τους θα έχουν γωνία ίση περίπου με 45^o.

Μια γωνία θέασης 45^o υποδηλώνει ότι η απόσταση του προβεβλημένου σημείου από το (0,0) είναι ίση με την απόσταση του σημείου (0,0,-k) από το (0,0). Το αποτέλεσμα θα είναι να μετακινηθούν όλα τα σημεία του επιπέδου xy θα μετακινηθούν στον κύκλο ακτίνας k που βρίσκεται επί του επιπέδου xy με κέντρο το (0,0,0). Στην εικόνα 3 μπορεί κανείς να παρατηρήσει κατά πόσο η απόσταση των σημείων από το κέντρο του κύκλου επηρεάζει την μετακίνηση των σημείων προς το κέντρο.

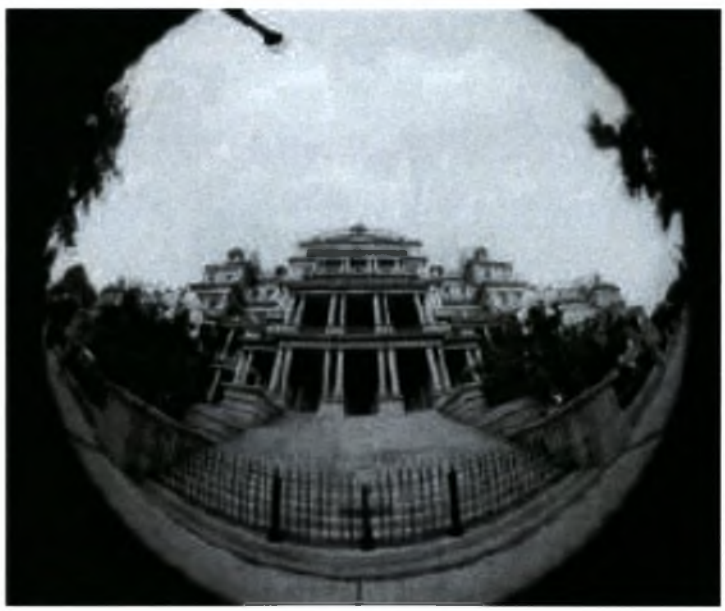


Εικόνα 3

Βέβαια είναι δυνατόν να τροποποιήσουμε αυτή την προβολή ώστε να καλύπτουμε χώρο παραπάνω από 180^o. Στην εικόνα 4 φαίνεται πώς γίνεται κάλυψη ενός χώρου 220^o λυγίζοντας το επίπεδο xy προς τα πίσω (για την ακρίβεια προς το αρνητικό άξονα του z) και προβάλλοντας τα τρισδιάστατα σημεία που βρίσκονται στο δεξί μέρος του λυγισμένου πεδίου. Μόλις τελειώσει αυτή η διαδικασία, τα σημεία θα είναι στον κύκλο ακτίνας k. Η εικόνα 5 είναι ένα παράδειγμα τέτοιας προβολής. Φαίνεται ότι τόσο οι κάθετες (δέντρο) όσο και οι οριζόντιες γραμμές (φράκτης) καμπυλώνουν και στοιχεία κοντά στο κέντρο της εικόνας είναι λεπτομερέστερα από αυτά που βρίσκονται στην περιφέρεια της εικόνας. Αυτός είναι και ο λόγος που αυτού του είδους η προβολή δεν χρησιμοποιείται εκτενώς.



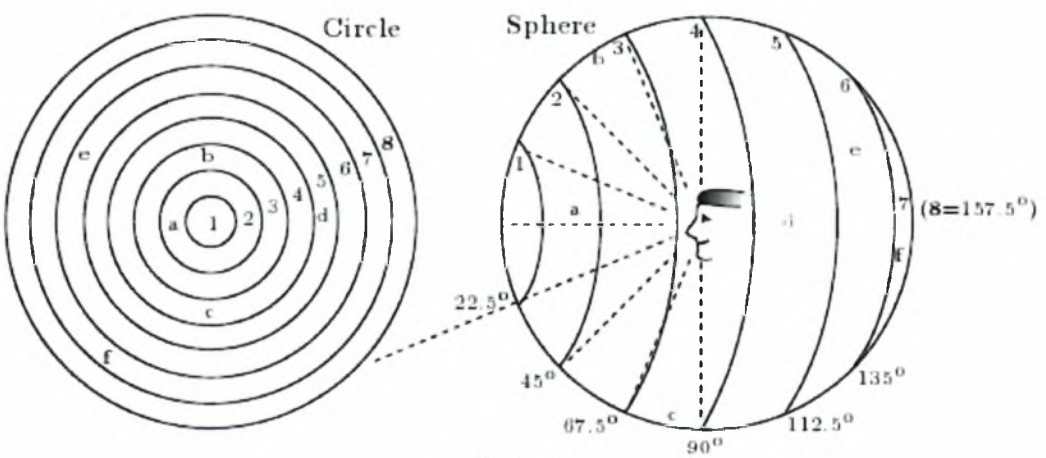
Εικόνα 4



Εικόνα 5

2.2 Angular Fisheye

Στο hemispherical fisheye δίνεται περισσότερη έμφαση στα σημεία που είναι κοντά στην γραμμή όρασης του παρατηρητή. Αυτά τα σημεία προβάλλονται με λεπτομέρεια ενώ τα μέρη που είναι οντά στην περιφέρεια της εικόνας προβάλλονται σε συμπιεσμένη μορφή κοντά στα όρια της προβολής. Αντίθετα στο angular fisheye δίνεται ίδια έμφαση σε όλα τα μέρη της εικόνας. Κάθε μέρος συμπιέζεται στον ίδιο βαθμό. Ένας καλύτερος ορισμός για την περιγραφή αυτής της προβολής θα ήταν «linear (γραμμικό) fisheye», αλλά ο όρος linear είναι αδόκιμος διότι ακόμα και σε αυτή την προβολή υπάρχουν παραμορφώσεις άρα είναι μη γραμμική. Ένα ιδιαίτερο χαρακτηριστικό αυτής της προβολής είναι ότι μπορεί να επεκταθεί για γωνίες θέασης άνω των 180^{o} και ακόμα να καλύψει τον χώρο 360^{0} που περιβάλλει τον παρατηρητή. Στην εικόνα 6 φαίνεται πώς γίνεται αυτό. Η (άπειρη) σφαίρα του χώρου που περιβάλλει τον παρατηρητή χωρίζεται σε 8 κάθετα κομμάτια ίσης γωνίας θέασης, κάθε ένα των οποίων προβάλλεται σε ένα δακτύλιο της τελικής κυκλικής προβολής. Για την ακρίβεια, βλέπουμε μόνο τα 7 των 8 κομματιών, διότι κοιτάζουμε την σφαίρα υπό γωνία.



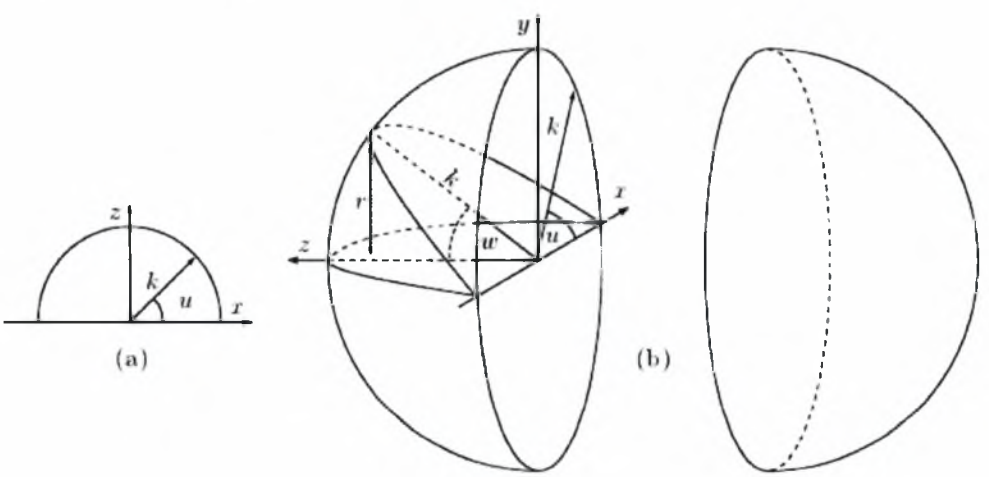
Εικόνα 6

Τα 6 σημεία a έως f φαίνονται στην σφαίρα με την προβολή τους στον κύκλο. Μπορεί κανείς να δει ότι το σημείο 'd' (στο 5_{o} κομμάτι) υποτίθεται πως βρίσκεται στη μεριά της σφαίρας μακριά από εμάς για αυτό προβάλλεται στη δεξιά μεριά του δακτυλίου 5.
Η μαθηματική ανάλυση της μεθόδου, αν και κουραστική, απαιτεί βασικές γνώσεις γεωμετρίας και τριγωνομετρίας. Κατ' αρχήν, παρατηρείστε στην εικόνα 6 ότι υπάρχει μια διακεκομμένη γραμμή. Λίγη σκέψη αρκεί να πείσει τον αναγνώστη ότι όλα τα σημεία στον χώρο κατά μήκος αυτής της γραμμής προβάλλονται στο ίδιο σημεία στον

κύκλο ακτίνας k. Άρα η δημιουργία ενός angular fisheye 360° γίνεται σαρώνοντας όλο τον χώρο γύρω από τον παρατηρητή και, για κάθε κατεύθυνση στο χώρο, επιλέγοντας το σημείο στη σκηνή που είναι το κοντινότατο στον παρατηρητή. Το σημείο αυτό θα προβληθεί στην επιφάνεια της σφαίρας και το σάρωμα θα συνεχιστεί στην επόμενη κατεύθυνση. Αφού έχουν εξεταστεί όλες οι κατευθύνσεις, η επιφάνεια της σφαίρας ακτίνας k γύρω από τον παρατηρητή θα είναι γεμάτη (προβεβλημένα) σημεία. Το επόμενο βήμα είναι να διαιρέσουμε την σφαίρα σε κομμάτια και να προβάλλουμε κάθε κομμάτι στον κύκλο ακτίνας k. Για την ακρίβεια, μπορούμε να θεωρήσουμε μια σφαίρα ακτίνας k με κέντρο τον παρατηρητή και να βρούμε πώς να σαρώσουμε την και να προβάλλουμε κάθε σημείο της σφαίρας στον κύκλο ακτίνας k. Στην εικόνα 7(a) έχουμε το ημικύκλιο ακτίνας k στο επίπεδο xz. Οι γνώστες των παραμετρικών αναπαραστάσεων γραμμών και επιφανειών γνωρίζουν ότι η παραμετρική αναπαράσταση του ημικυκλίου είναι k(cos*u*,sin*u*) όπου 0≤u≤180°. Μια πλήρης σφαίρα ακτίνας 360° δημιουργείται με την περιστροφή του παραπάνω ημικυκλίου κατά 360° μοίρες γύρω από τον άξονα x. Άρα, η παραμετρική εξίσωση της σφαίρας είναι το γινόμενο της παραμετρικής εξίσωσης του ημικυκλίου με τον πίνακα περιστροφής γύρω από τον άξονα x,

$$k(\cos u, 0, \sin u)\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos w & -\sin w \\ 0 & \sin w & \cos w \end{pmatrix} = k(\cos u, \sin u \sin w, \sin u \cos w),$$

για 0≤u≤180° και 0≤w≤360°.



Εικόνα 7

Η λέξη «βαρυκεντρικός» προέρχεται από την λέξη «βαρύκεντρο» δηλαδή κέντρο βαρύτητας, διότι τέτοια βάρη χρησιμοποιούνται για τον υπολογισμό του κέντρου βαρύτητας ενός αντικειμένου. Τα βαρυκεντρικά βάρη έχουν πολλές εφαρμογές στην γεωμετρία γενικότερα και ειδικά στον σχεδιασμό γραμμών και επιφανειών. Στην εικόνα 7(b) το ημικύκλιο στο xz επίπεδο και πώς περιστρέφεται. Είναι φανερό ότι η γωνία w ενός σημείου **P** στην σφαίρα είναι μια από τις παραμέτρους του προβαλλομένου σημείου P^*. Αυτή η γωνία καθορίζει την απόσταση r του σημείου P^* από το κέντρο του κύκλου ακτίνας k. Γενικά το r ισούται με k*sin*w*, αλλά το θέμα είναι ότι για w=0

θέλουμε r=0, ενώ για w=90° θέλουμε r=k/2 και όχι r=k. Αυτό γίνεται διότι οι τιμές του r που κυμαίνονται από k/2 έως k αντιστοιχούν στο δεξί ημισφαίριο (δηλαδή από 90° έως 270°). Άρα για τιμές του w από 0 έως 90° γράφουμε ότι r=(k/2)*sin*w*, και ο πίνακας 1 περιέχει όλες τις εκφράσεις του r για τα εναπομείναντα διαστήματα του w. Αφού έχουμε υπολογίσει το r, μας μένει να υπολογίσουμε που να τοποθετήσουμε το σημείο P^* στον κύκλο ακτίνας k, πράγμα το οποίο καθορίζεται από το *u*. Αυτή η μεταβλητή ανήκει στο διάστημα και το σημείο P^* θα τοποθετηθεί είτε στο άνω ήμισυ του κύκλου (αν 0≤w≤180°) είτε στο κάτω ήμισυ του κύκλου (180°≤w≤360°) όπως φαίνεται στον πίνακα 1.

w	r	r interval	u	$\sin w$
$0 \rightarrow 90$	$\frac{k}{2} \sin w$	$[0, k/2]$	top	$0 \rightarrow 1$
$90 \rightarrow 180$	$(1 - \frac{\sin w}{2})k$	$[k/2, k]$	top	$1 \rightarrow 0$
$180 \rightarrow 270$	$(1 + \frac{\sin w}{2})k$	$[k, k/2]$	bottom	$0 \rightarrow -1$
$270 \rightarrow 360$	$-\frac{k}{2} \sin w$	$[k/2, 0]$	bottom	$-1 \rightarrow 0$

Πίνακας 1

Η πλήρης απεικόνιση της σφαίρας ακτίνας k στον κύκλο ακτίνας k γίνεται σε διπλό βρόχο, όπου το w ποικίλει σε τιμή από 0 έως 360° στον εξωτερικό βρόχο ενώ το u ποικίλει από 0 έως 180° στον εσωτερικό βρόχο. Για κάθε ζεύγος (u,w) το σημείο της τρισδιάστατης σκηνής που είναι πιο κοντά στον χρήστη (ο οποίος με την σειρά του είναι στο κέντρο) καθορίζεται και προβάλλεται υπολογίζοντας την τιμή r από τον πίνακα 1 και χρησιμοποιώντας το ζεύγος (r,u) τόσο ως πληροφορία για το "up" ή το "down" όσο και ως πολικές συντεταγμένες του P^*.

w	r	r interval	u	$\sin w$
$0 \rightarrow 90$	$k \sin w$	$[0, k]$	top	$0 \rightarrow 1$
$270 \rightarrow 360$	$-k \sin w$	$[k, 0]$	bottom	$-1 \rightarrow 0$

Πίνακας για fisheye 180°

Το σημείο πίσω από τον παρατηρητή αποτελεί μια ειδική περίπτωση. Το σημείο αυτό το φτάνουμε όταν w=180° (δηλαδή r=k), στην οποία περίπτωση κάθε τιμή του u θα επιλέξει αυτό το σημείο. Αυτό το ειδικό σημείο θα απεικονισθεί σε κάθε σημείο του κύκλου r=k. Συχνά μία τρισδιάστατη σκηνή κατέχει κάθε διεύθυνση στο χώρο. Η σκηνή μπορεί να αποτελείται από πολλά αντικείμενα με κομμάτια εδάφους, νερού και ουρανού να γεμίζουν κάθε σημείο. Σε τέτοιες περιπτώσεις, κάθε διεύθυνση (u,w) θα αντιστοιχεί σε τουλάχιστον ένα σημείο της σκηνής. Μερικές φορές μια σκηνή αποτελείται μόνο από αντικείμενα άνευ χρωματικού υπόβαθρου. Σ' αυτές τις περιπτώσεις πολλά ζεύγη (u,w) δεν θα αντιστοιχούν σε κανένα σημείο της σκηνής. Για ένα τέτοιο ζεύγος, η προβολή του

στο κύκλο ακτίνας k μπορεί να έχει χρώμα άσπρο ή οποιοδήποτε άλλο χρώμα υπόβαθρου.
Όταν όλος ο χώρος γύρω από τον παρατηρητή προβάλλεται σε έναν κύκλο, τότε η προβολή angular fisheye γίνεται ένας από τους πολλούς τρόπους για την απεικόνιση μιας σφαίρας σε ένα επίπεδο. Κάθε προβολή σφαίρας σε επίπεδο εισάγει παραμορφώσεις και οι 2 κύριες παραμορφώσεις της προβολής angular fisheye είναι ότι:

- οι ευθεία γραμμές απεικονίζονται ως καμπύλες και
- το ημισφαίριο μπροστά του παρατηρητή προβάλλεται στο εσωτερικό ήμισυ του κύκλου και, με λίγη προσπάθεια, μπορεί να γίνει κατανοητό, αλλά το ημισφαίριο πίσω από τον παρατηρητή προβάλλεται στο εξώτερο ήμισυ του κύκλου, που είναι ένας δακτύλιος, πράγμα που καθιστά δύσκολη την κατανόηση των λεπτομερειών του.

Στην εικόνα 8 έχουμε 2 παραδείγματα προβολής angular fisheye 180^{o}.

Εικόνα 8

Είναι εύκολο να δει κανείς ότι η παραμόρφωση είναι ομοιόμορφη σε όλη την εικόνα. Επίσης παρόλο που οι ίσιες γραμμές καμπυλώνονται, είναι εμφανές ότι η κυρτότητα μειώνεται σε γραμμές που είναι κοντά στο κέντρο της εικόνας. Στην εικόνα 8 (αριστερά) τόσο οι κάθετες (τα δέντρα) όσο και οι οριζόντιες (γραμμή του ορίζοντα και τα καθίσματα) γραμμές καμπυλώνουν και οι λεπτομέρειες της εικόνας γίνονται μεγαλύτερες όσο πάμε στο κέντρο.
Κάτι άλλο που πρέπει να επισημανθεί είναι ότι η σφαίρα είναι μεγαλύτερη από τον κύκλο. Ακόμα και αν οι μεταβλητές u και w εξετάζονται ανά μεγάλα βήματα, είναι πολύ πιθανόν να υπάρχουν περισσότερες διευθύνσεις να σαρωθούν από όσα είναι τα πίξελ στον κύκλο ακτίνας k. Άρα χρειαζόμαστε μία άλλη προσέγγιση για να δημιουργήσουμε μια προβολή angular fisheye. Αντί να σαρώνουμε την 360^{o} σφαίρα σε πολλές διευθύνσεις, σαρώνουμε τον κύκλο ακτίνας k ανά πίξελ, υπολογίζουμε τις πολικές

συντεταγμένες (r,u) κάθε πίξελ και μετά τις χρησιμοποιούμε για να καθορίσουμε την αντίστοιχη διεύθυνση (u,w) στον χώρο. Αν ένα σημείο της σκηνής βρίσκεται σε αυτή την διεύθυνση, τότε προβάλλεται στο πίξελ άνευ περαιτέρω υπολογισμών.
Πιο αναλυτικά, θεωρούμε ότι ο κύκλος είναι "ενσωματωμένος" σε ένα ορθογώνιο bitmap ύψους H pixel και πλάτους W pixel. Σαρώνουμε το ορθογώνιο ανά στήλη. Αν το τρέχον pixel έχει συντεταγμένες (a,b), τότε τις μετατρέπουμε σε κανονικοποιήμενες συντεταγμένες (x,y) στο διάστημα [-k,+k] με την εξής διαδικασία:

$$x = \left(\frac{2a}{W} - 1\right) k \text{ and } y = \left(\frac{2b}{H} - 1\right) k$$

Η απόσταση του pixel από το κέντρο της ορθογώνιας εικόνας είναι $r = \sqrt{x^2 + y^2}$

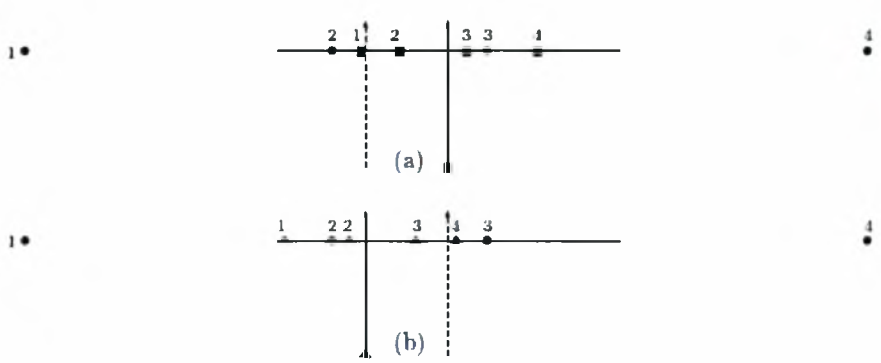
Αν r είναι μεγαλύτερο του k, τότε το σημείο είναι εκτός του κύκλου ακτίνας k οπότε αγνοείται. Ειδάλλως, η γωνία u υπολογίζεται ως εξής:

$$u = \begin{cases} 0, & r = 0, \\ \pi - \arcsin(y/r), & x < 0, \\ \arcsin(y/r), & x \geq 0 \end{cases}$$

Η γωνία w ισούται με r/2, άρα έχει πεδίο τιμών το διάστημα [0, k/2] και το διάνυσμα διεύθυνσης είναι k(cos*u*, sin*u**sin*w*, sin*u**cos*w*).
Η εισγώμενη απο το angular fisheye παραμόρφωση μπορεί να χρησιμοποιηθεί να αποκτήσουμε μια σφαιρική πανοραμική προβολή. Φανταστείτε έναν κύκλο ακτίνας κστο οποίο έχουμε προβάλλει ένα angular fisheye 180°. Σαρώνουμε τον κύκλο ανα pixel και μετατρέπουμε τις Καρτεσιανές συντεταγμένες (a,b) των pixel σε πολικές συντεταγμένες $r=\sqrt{a*a+b*b}$ και u=arctan (b/a) (αν a=0, τότε, ανάλογα με την τιμή του b, u=0 ή u=180°). Αφού βρούμε το r, χρησιμοποιούμε την σχέση r=±k*sin*w* για να υπολογίσουμε την γωνία w. Αφού υπολογίσαμε τις γωνίες u και w, γνωρίζουμε ότι το pixel (a,b) είναι η προβολή ενός σημείου P ευρισκόμενου στην διεύθυνση (cos*u*, sin*u**sin*w*, sin*u**cos*w*) στο ημισφαίρια ακτίνας k με κέντρο τον παρατηρητή. Άρα στην θεωρία είναι δυνατόν να αντιστοιχίσουμε κάθε pixel στο angular fisheye σε ένα τρισδιάστατο σημείο P στο ημισφαίριο. Δεν γνωρίζουμε πόσο μακριά ήταν από τον παρατηρητή το αρχικό σημείο, διότι αυτή η πληροφορία χάθηκε, όταν η προβολή fisheye δημιουργήθηκε, αλλά γνωρίζουμε ότι από όλα τα τρισδιάστατα σημεία στην διεύθυνση (u,w) το σημείο P ήταν το κοντινότατο στον παρατηρητή, αποκλείοντας όλα τα άλλα σημεία που βρίσκονται πίσω από αυτό. Στην πράξη όμως, είναι δύσκολο να υλοποιηθεί αυτή η τεχνική διότι ο αριθμός των pixel του κύκλου είναι κατά πολύ μικρότερος του αριθμού των pixel στο ημισφαίριο.

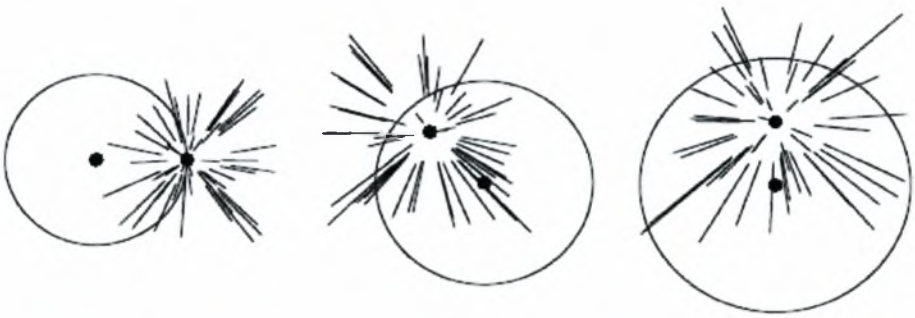
2.3 Off-axis Fisheye

Στα προαναφερθέντα είδη προβολής fisheye υποθέταμε ότι ο παρατηρητής βλέπει μια σφαίρα επί της οποίας προβάλλεται ένα ημισφαίριο άπειρης ακτίνας.
Στις δε εικόνες 2 και 6 υπονοείται ότι η γραμμή του παρατηρητή περνά μέσα από το κέντρο του κύκλου. Μπορεί να ειπωθεί ότι ο παρατηρητής βρίσκεται πάνω στον άξονα του κύκλου και μπορούμε να αναρωτηθούμε τι θα δει ο παρατηρητής όταν μετακινηθεί μακριά από τον άξονα, κοιτάζοντας προς την ίδια διεύθυνση. Αυτό δεν είναι ένα θεωρητικό πρόβλημα. Πολλά πλανητάρια χρησιμοποιούν φακούς fisheye για την προβολή μιας εικόνας επί του ημισφαιρικού θόλου, όπου κάποιοι (ίσως και πολλοί) παρατηρητές βρίσκονται μακριά από το κέντρο. Οι παρατηρητές αυτοί βλέπουν μία διπλά «παραμορφωμένη» εικόνα αφ' ενός επειδή είναι η προβολή είναι fisheye, αφ' ετέρου επειδή την παρατηρούν όντας εκτός του άξονα (εξ ου και off-axis). Τα μαθηματικά μιας off-axis fisheye προβολής παρουσιάζονται στην εικόνα 9. Ξεκινούμε με 4 σημεία, απεικονισμένα ως κύκλοι και αριθμημένα από 1 έως 4.



Εικόνα 9

Στο πρώτο μέρος (a) της εικόνας, θεωρείται ότι ο παρατηρητής είναι επί του άξονα και τα σημεία μετακινούνται προς τον παρατηρητή μειώνοντας στο μισό τις γωνίες θέασης. Τα μετακινούμενα σημεία απεικονίζονται ως μικρά τετράγωνα. Στο μέρος (b), θεωρείται ότι ο παρατηρητής βρίσκεται εκτός του άξονα και ότι τα 4 σημεία μετακινούνται προς τον παρατηρητή μειώνοντας στο μισό τις γωνίες θέασης. Τα νέα σημεία απεικονίζονται ως μικρά τρίγωνα. Είναι εμφανές ότι τα σημεία 1 και 2 μετακινούνται περισσότερο στο μέρος (a) από ότι στο μέρος (b). Άρα, αυτά τα μέρη της εικόνας παραμορφώνονται περισσότερο όταν ο παρατηρητής είναι επί του άξονα. Αντίθετα, τα σημεία 3 και 4 μετακινούνται περισσότερο όταν ο παρατηρητής είναι εκτός άξονα, οπότε παραμορφώνονται τα μέρη της εικόνας στο δεξί μέρος. Στην εικόνα 10 παρουσιάζεται γενικότερα το αποτέλεσμα μιας προβολής off-axis. Δείχνει 50 σημεία μετακινούμενα προς έναν εκτός άξονα παρατηρητή. Στα 3 μέρη της εικόνας, από αριστερά προς δεξιά, ο παρατηρητής είναι τοποθετημένος στα



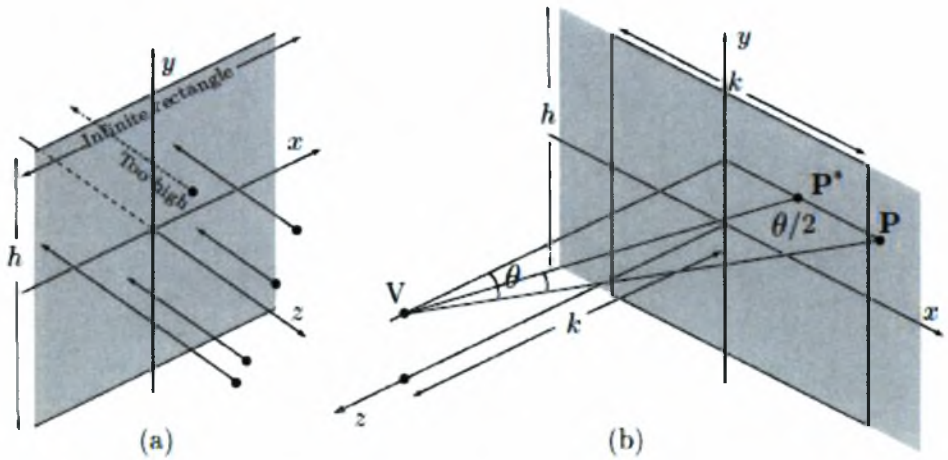
```
k = 10; n = 50; scal[q_] := (k Tan[ArcTan[q/k]/2])/q;
P = Table[{Random[Real, {-10.,10.}], Random[Real, {-10.,10.}]}, {n}];
x = -5; y = 5; (* Location of viewer *)
Pt = P - Table[{x, y}, {n}];
Q = Table[Sqrt[Pt[[i]].Pt[[i]]], {i, n}];
L = Table[Line[{P[[i]]+{x, y}, (scal[Q[[i]]] P[[i]])+{x, y}}], {i, n}];
Show[Graphics[L], Graphics[Circle[{0, 0}, k]],
 Graphics[{AbsolutePointSize[5], Point[{0, 0}]}],
 Graphics[{AbsolutePointSize[5], Point[{x, y}]}],
 AspectRatio -> Automatic, PlotRange -> All]
```

Εικόνα 10

σημεία (10,0), (-5,5) και (0,5). Η εικόνα δείχνει τι γίνεται όταν ο παρατηρητής είναι εκτός άξονα και αγνοεί τις παραμορφώσεις (όπως οι ίδιες γραμμές που παραμορφώνονται και γίνονται καμπύλες) που εισάγει το fisheye.

2.4 Rectangular Fisheye Projection

Η προβολή hemispherical fisheye προβάλλει έναν χώρο 180° που βρίσκεται μπροστά από τον παρατηρητή, μια απείρως μεγάλη εικόνα, σε πεπερασμένου μεγέθους κύκλο, και αυτό το κάνει παραμορφώνοντας την εικόνα, ειδικά στις περιοχές που είναι μακριά από το κέντρο του. Η προβολή rectangular fisheye λειτουργεί όπως η προβολή hemispherical fisheye αλλά σε διαφορετικό βαθμό. Δημιουργεί λιγότερη παραμόρφωση αλλά μπορεί να προβάλλει μόνο ένα μέρος του χώρου που βρίσκεται μπροστά από τον παρατηρητή. Τα μέρη που είναι πολύ ψηλά από τον παρατηρητή ή πολύ χαμηλά δεν συμπεριλαμβάνονται σε αυτόν τον τύπο προβολής. Στην εικόνα 11(a) δείχνεται η βασική σκέψη. Έστω ένα ορθογώνιο απείρου πλάτους και πεπερασμένου ύψους h τοποθετημένο στο επίπεδο xy. Ένα τρισδιάστατο σημείο (x, y, z) προβάλλεται, πάνω στο επίπεδο, στο σημείο (x, y, 0), αν και μόνο αν η συντεταγμένη y ανήκει στο διάστημα $[\ -h/2\ ,+\ h/2\ ]$. Στην εικόνα φαίνεται ένα σημείο που είναι πολύ ψηλά. Σημεία που είναι άνω ή κάτω του ορθογωνίου δεν συμπεριλαμβάνονται στην προβολή.

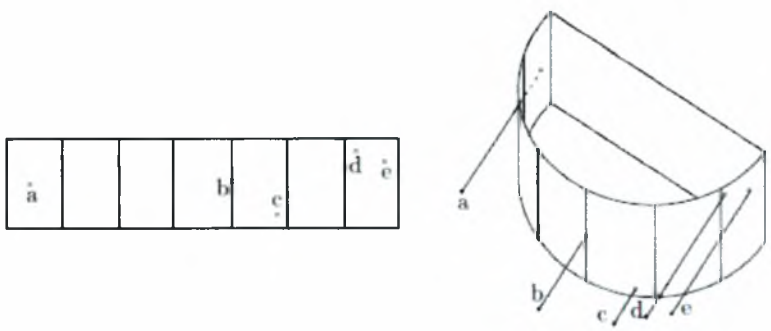


Εικόνα 11

Μόλις ένα σημείο προβληθεί στο ορθογώνιο, μετακινείται στην διεύθυνση x, για να το φέρουμε μέσα στο ορθογώνιο πλάτους k. Αυτό γίνεται μειώνοντας στο μισό την γωνία θέασης θ, όπως και στην προβολή hemispherical fisheye, αλλά μόνο στην διεύθυνση του x όπως βλέπουμε στην εικόνα 11(b). Η τελική προβολή παραμορφώνεται μόνο στην διεύθυνση x. Όλα τα y διατηρούνται ως έχουν. Το τελικό αποτέλεσμα είναι η προβολή του σημείου (x, y, z) στο σημείο (s*x, y, 0) όπου s υπολογίζεται από την ακόλουθη εξίσωση:

$$s = \frac{k \tan((\arctan[|x|/k])/2)}{|x|}.$$

Αυτή η παραλλαγή της προβολής fisheye είναι συγγενής της προβολής semicylindrical fisheye. Ξεκινούμε με έναν μισό κύλινδρο, επί της επιφάνειας του οποίου τρισδιάστατα σημεία προβάλλονται σε παράλληλη διεύθυνση. Τότε ο μισός κύλινδρος ξετυλίγεται και θεωρείται ένα επίπεδο ορθογώνιο. Παρατηρείστε ότι τα σημεία «e» και «d» στην εικόνα 12 είναι κοντά στον



Εικόνα 12

τρισδιάστατο χώρο αλλά οι προβολές τους στον κύλινδρο είναι έχουν μεγαλύτερη μεταξύ τους απόσταση. Αυτού του είδους η προβολή μεγεθύνει λεπτομέρειες κοντά στις κάθετες ακμές της τελικής προβολής, που είναι το αντίθετο για τις άλλες παραλλαγές της προβολής fisheye.

3 Cube map

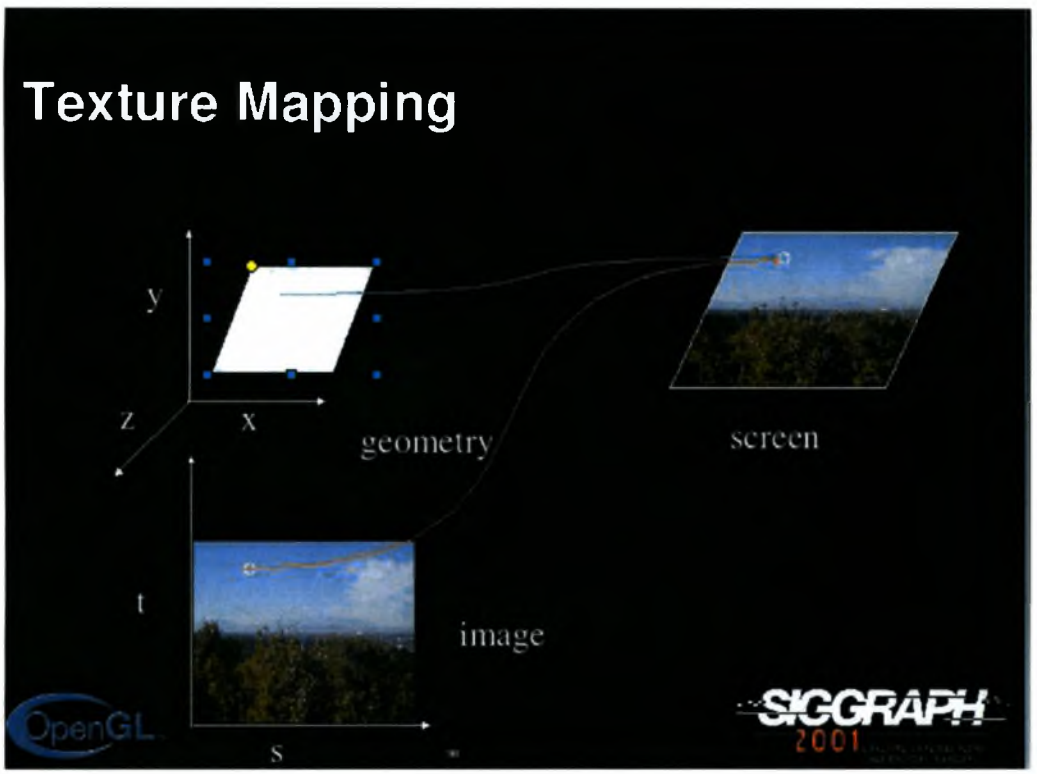
Όπως είδαμε και στην περίληψη, στόχος στης διπλωματικής είναι να μετατρέψουμε ένα cube map σε fisheye.

Cubemap Fisheye

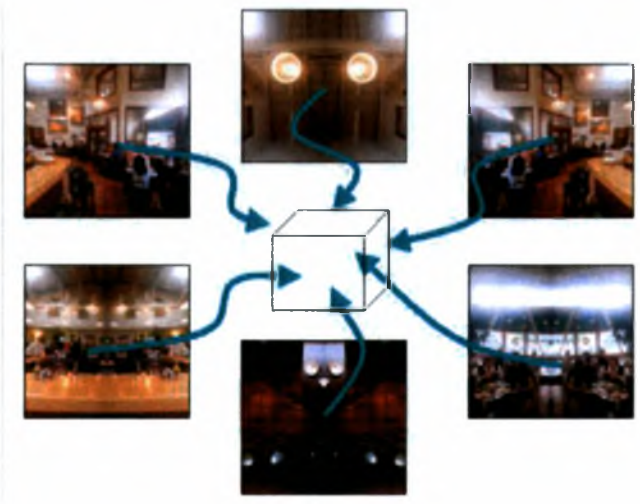
Το cubemap είναι ένα texture που αναπαριστά έναν χώρο 360° που περιβάλλει μία περιοχή ή ένα αντικείμενο. Πιο αναλυτικά, είναι ένα σύνολο 6 ξεχωριστών κομματιών textures τα οποία είναι τοποθετημένα στις πλευρές ενός φανταστικού κύβου. Χρησιμοποιείται πάρα πολύ σε διάφορες εφαρμογές απεικόνισης περιβάλλοντος (environment mapping), dynamic cubemap reflections, Skylight Illumination, Stable Specular Highlights καθώς και για Fancy Per-Pixel Lighting. Παρακάτω θα εξηγηθεί μόνο το environment mapping, πάνω στο οποίο βασίζεται το θέμα της διπλωματικής

3.1 Environment Mapping

Πολλές φορές είναι προτιμότερο, για λόγους ευκολίας, αντί να «ζωγραφίζουμε» μια εικόνα σε ένα συγκεκριμένο σχήμα, να απεικονίζουμε σε αυτό σχήμα αυτήν την εικόνα.

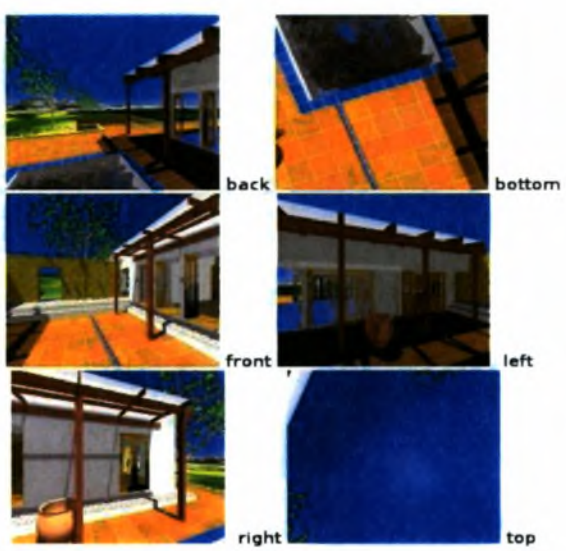


Η απεικόνιση (mapping) πολλές φορές είναι προτιμότερη όταν έχουμε να κάνουμε με τρισδιάστατους χώρους. Σε αυτή την περίπτωση μιλάμε για environment mapping. Ένας τρόπος απεικόνισης περιβάλλοντος είναι με την χρήση μίας μεθόδου, γνωστή ως Cube map texturing! Αυτή η μέθοδος είναι μία μέθοδος απεικόνισης υφής (texture mapping) που χρησιμοποιεί ένα διάνυσμα 3 διαστάσεων για την απεικόνιση 6 δυσδιάστατων υφών, που είναι διατεταγμένες όπως οι πλευρές ενός κύβου, σε ένα οποιοδήποτε σχήμα. Φανταστείτε το περιβάλλον γύρω σας. Μπορείτε να κάνετε capture τον περιβάλλοντα χώρο 360^{o}, όντας σε ένα μέρος και φωτογραφίζοντας 6 εικόνες, κάθε με διαφορά 90^{o} μοίρες από τις άλλες.



Cubemap

Αυτή η μέθοδος απεικόνισης είναι hardware_accelerated, δηλαδή είναι άμεσα υποστηριζόμενη από την κάρτα γραφικών του υπολογιστή. Έτσι είναι δυνατόν να κάνεις render ένα δυναμικό αντικείμενο που αντανακλά ένα περιβάλλον. Αυτό μπορεί να γίνει σε πραγματικό χρόνο.



Cubemap2

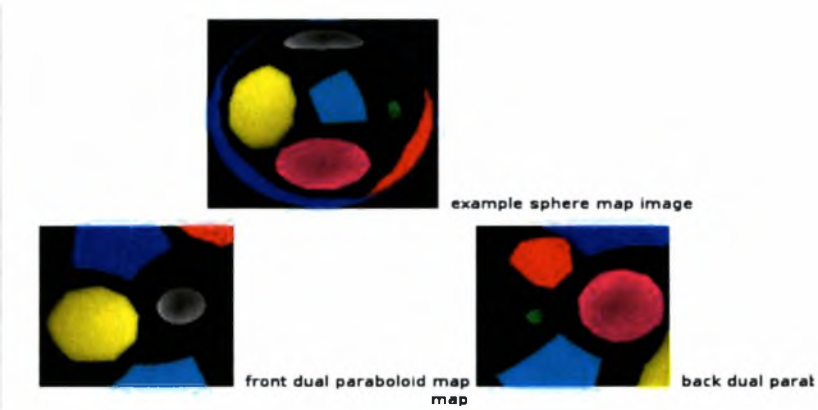
Παρακάτω βλέπουμε το αποτέλεσμα του cube map texturing, που έγινε με βάση το cube map στην προηγούμενη εικόνα, σε υπολογιστή που χρησιμοποιεί την νέα GeForce 256 GPU της NVIDIA.

Cube map texturing

Σε αυτήν την εικόνα βλέπουμε την απεικόνιση περιβάλλοντος σε μια φούσκα της οποίας η επιφάνεια μεταβάλλεται συνεχώς.
Βέβαια το cube map texturing δεν είναι η μόνη μέθοδος για environment mapping. Άλλες προσεγγίσεις όπως το sphere mapping και το dual paraboloid mapping μπορούν να παράγουν παρόμοια αποτελέσματα χρησιμοποιώντας το συμβατικό texture mapping, αλλά εμφανίζουν σοβαρά μειονεκτήματα που περιορίζουν την γενική χρήση τους. Το sphere mapping είναι απλό και άμεσα υποστηριζόμενο από την OpenGL, αλλά είναι view_dependent, υποδηλώνοντας ότι για κάθε διαφορετικό eye position χρειαζόμαστε και μια διαφορετική εικόνα υφής για το sphere map (sphere map texture image). Επιπλέον, το sphere mapping εμφανίζει συχνά στιγματοειδή αντικείμενα στις παρυφές των αντικειμένων με sphere map. Παρακάτω μια εικόνα δείχνει αυτά τα στιγματοειδή αντικείμενα.

Sphere mapping

Το dual paraboloid mapping απαλείφει αυτά τα λάθη, αλλά προγραμματιστικά είναι ακριβό, διότι απαιτεί 2 texture units ή 2 render passes. Επιπλέον απαιτεί περίπλοκα μαθηματικά για involved texture coordinate generation (εκτός και αν υπάρχει ειδικό extension για OpenGL texgen reflection). Είτε πρόκειται για sphere maps είτε για dual paraboloid maps απαιτούνται ειδικές πράξεις για image warping. Παραδείγματα της απαραίτητης στρέβλωσης για το sphere map και για το dual paraboloid map φαίνονται παρακάτω.



Το Cube map texturing είναι απαλλαγμένο από τα στίγματα που εμφανίζονται στο sphere mapping. Σε αντίθεση με το dual paraboloid, απαιτείται ένα μόνο texture unit και μόνο

ένα πέρασμα (render pass). Επειδή οι εικόνες υφής (texture images) για το cube map είναι απλά οι 6 κυβικές πλευρές ενός cube environment, οι υφές των cube map είναι ευκολότερο να αποκτηθούν από φωτογράφους ή να δημιουργηθούν δυναμικά. Επίσης, χρησιμοποιεί πλήρως την ανάλυση (resolution) όλης της εικόνας. Η σφαίρα και τα paraboloid map χρησιμοποιούν μόνο το 78% της συνολικής ανάλυσης του διαθέσιμου texture από το sphere και dual paraboloid map τα οποία απαιτούν επιπρόσθετα βήματα στρέβλωσης.
Από την άλλη, το cube map texturing είναι πιο απλό από την προσέγγιση των στρεβλωμένων sphere map ή dual paraboloid map, αλλά απαιτεί την ταυτόχρονη πρόσβαση στις 6 εικόνες υφής (texture images). Αυτό απαιτεί περιπλοκότερο texture hardware. Ενώ το cube map texturing είναι περιπλοκότερο της συμβατικής απεικόνισης υφής (texture mapping), ο αλματώδης ρυθμός της σμίκρυνσης των ημιαγωγών έχει καταστήσει τις single-chip implementations για cube map texturing μέσω hardware πραγματικότητα για τους σημερινούς καταναλωτές. Η σειρά GeForce 256 GPU της NVIDIA είναι η εμπορική σειρά GPU που υποστηρίζει το cube map texturing.
Το υλικό (hardware) είναι χρήσιμο μόνο, εάν οι διεπαφές του λογισμικού είναι κατάλληλες για την αξιοποίηση των δυνατοτήτων του υλικού. Ευτυχώς στην περίπτωση του texture cube mapping, όλες τα κύρια API για τρισδιάστατα γραφικά υποστηρίζουν το cube map texturing. Της Microsoft το Direct3d Api για τα Windows Pc υποστηρίζουν υφές cube map με την αναβάθμιση του DirectX 7. Η OpenGL, υποστηρίζει multi-vendor EXT_texture_cube_map extension για cube map texturing. Βέβαια, όπως προειπώθηκε, παρά την αποδοτικότητά τους, οι τελευταίες γενεές των GPU υποστηρίζουν cube map textures. Οι προσεγγίσεις sphere map και dual paraboloid map χρησιμοποιούνται ακόμη στην περίπτωση που η GPU δεν υποστηρίζει το cube map texturing. Στην διπλωματική χρησιμοποιείται για αυτό τον σκοπό το cube map texturing, παρακάτω όμως παρουσιάζονται συνοπτικά και άλλες χρησιμότητες του cube map.

4 Παρουσίαση του Κώδικα

Σε αυτό το σημείο, γίνεται η παρουσίαση και η επεξήγηση της λειτουργίας του κώδικα.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <gl/glut.h>
//#include<GL/glext.h>
#include <gl/gl.h>
#include<gl/glui.h>
#include "tga.h"

/* In case your <GL/gl.h> does not advertise EXT_texture_cube_map... */
#ifndef GL_EXT_texture_cube_map
# define GL_NORMAL_MAP_EXT              0x8511
# define GL_REFLECTION_MAP_EXT          0x8512
# define GL_TEXTURE_CUBE_MAP_EXT        0x8513
# define GL_TEXTURE_BINDING_CUBE_MAP_EXT    0x8514
# define GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT  0x8515
# define GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT  0x8516
# define GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT  0x8517
# define GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT  0x8518
# define GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT  0x8519
# define GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT  0x851A
# define GL_PROXY_TEXTURE_CUBE_MAP_EXT      0x851B
# define GL_MAX_CUBE_MAP_TEXTURE_SIZE_EXT   0x851C
#endif

/* constants */
#ifndef M_PIPositive
#define M_PI          (3.141592653589793f)
#endif
#define DTOR          (M_PI/180.0)
#define RTOD          (180.0/M_PI)

#define CUBE_MAP_SIZE 256

#define ZTRANS 4.0F



gliGenericImage *image[6],*image2[6];
enum {CUBE_POS_X, CUBE_NEG_X, CUBE_POS_Y, CUBE_NEG_Y, CUBE_POS_Z, CUBE_NEG_Z};
unsigned char *data2[6];
//char
*fileName[6]={"cm_left.tga","cm_right.tga","cm_top.tga","cm_bottom.tga","cm_back.tga","cm_front.tga"},*fileName2[6];
char *fileName2[6];
float r=1.0f,aperture=M_PI/2.0,rx=0.0,ry=0.0,rz=0.0;
GLuint TexID = 0;
```

```
int fullscreen = 0,w=0,h=0,main_window,selection=0;


struct point{
        double x,y,z;
};

GLUI *glui_subwin;
GLUI_Spinner *aprtr, *radius;
GLUI_Button *reset,*end,*rotator,*new_Fish_Eye;
GLUI_Panel *fisheye_parameters,*rotation,*cube_faces;
GLUI_Checkbox *x_axis,*y_axis,*z_axis,*count_clock;
GLUI_EditText *faces[6];

//typedef enum {CUBE_MAP,FISH_EYE} ProjecType;
//ProjecType gViewType=CUBE_MAP;


enum {APERTURE_SPIN, RADIUS_SPIN,RESET, EXIT,X,Y,Z,R,CCW, NX,PX,NY,PY,NZ,PZ,NEW};

GLenum cubefaces[6] = {
  GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT,
  GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT,
  GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT,
  GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT,
  GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT,
};

//Set Direction Vector for each point in fisheye and draw
void vert(float *theta, float *phi)
{

  float x[4], y[4], z[4], nx[4], ny[4], nz[4],r1[4];
  int k;
  for(k=0;k<4;k++){

        nx[k] = r*cos(DTOR * theta[k]) * sin(phi[k]*aperture/2*DTOR);
        ny[k] = r*sin(DTOR * theta[k]) * sin(phi[k]*aperture/2*DTOR);
        r1[k] = sqrt(nx[k]*nx[k]+ny[k]*ny[k]);

        nx[k] = r*cos(DTOR * theta[k]) * sin((phi[k]*aperture/2)*DTOR);
        ny[k] = r*sin(DTOR * theta[k]) * sin((phi[k]*aperture/2)*DTOR);
        nz[k]=r*cos((r1[k]*aperture/2)*DTOR);
        glNormal3f(nx[k], ny[k], nz[k]);

        x[k] = r * cos(DTOR * theta[k]) * sin(DTOR * phi[k]);
        y[k] = r * sin(DTOR * theta[k]) * sin(DTOR * phi[k]);
        z[k] =  -1.5*ZTRANS+r * cos(DTOR * phi[k]);
        glVertex4f(x[k], y[k], z[k], 1.0);


  }
}

//Find if the file with name "*c" is .tga format
```

```
int ext(const char *c){
        char *e=".tga";
        int l=strlen(c),i,k;
        k=l-strlen(e);
        for(i=k;i<l;i++){
                if(c[i]!=e[i-k]){
                        return 0;
                }
        }
        return 1;

}

//Draw the sphere
void DrawSphere()
{

        float scale,phi[4],theta[4],y,r[4];
        struct point p[4];
        int N=100,i,j,k;
        glDisable(GL_TEXTURE_2D);
        glColor4f(1.0, 1.0, 1.0, 1.0);
        scale=M_PI/N;
        glBegin(GL_QUAD_STRIP);
        for (j=-N;j<N;j++) {
                for (i=-N/2;i<N/2;i++) {
                        theta[0] = j * scale;
                        theta[1] = (j+1) * scale;
                        theta[2] = theta[1];
                        theta[3] = theta[0];
                        phi[0] = i * 2 * scale;
                        phi[1] = phi[0];
                        phi[2] = (i+1) * 2 * scale;
                        phi[3] = phi[2];

                        for (k=0;k<4;k++) {
                                y = sin(theta[k]);
                                p[k].x = y * sin(phi[k]);
                                p[k].y = y * cos(phi[k]);
                           p[k].z = cos(theta[k]);
                        }
                        for (k=0;k<4;k++)
                                r[k] = p[k].x*p[k].x+p[k].y*p[k].y;
    if (r[0] > 1 || r[1] > 1 ||
       r[2] > 1 || r[3] > 1)
      continue;
            for (k=0;k<4;k++){
                        phi[k]*=RTOD;
                        theta[k]*=RTOD;

            }
      vert(phi,theta);
                        }


        }
```

```
        glEnd();
        //glutSwapBuffers();
}

//Create the angular fisheye
void
Setup(gliGenericImage *im[])
{
  int i;
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(360*atan2(1.5,2*ZTRANS -4.0)/M_PI,4.0/3.0,2*ZTRANS -4.0, 2*ZTRANS + 2.0); // match
                                                // 640x480

  glMatrixMode(GL_MODELVIEW);;
  glLoadIdentity();
  gluLookAt(0.0, 0.0, 1.0,  /* eye is at (0,0,5) */
                        0.0, 0.0, 0.0,      /* center is at (0,0,0) */
                        0.0, 1.0, 0.);


  glEnable(GL_DEPTH_TEST);

  glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

  for (i = 0; i < 6; i++) {
    glTexImage2D(
      cubefaces[i],
      0,                // level
      im[i]->components,          // internal format
      im[i]->width,     // width
      im[i]->height,     // height
      0,                // border
      im[i]->format,           // format
      GL_UNSIGNED_BYTE,   // type
      im[i]->pixels); // pixel data
        //printf("i map\n");
  }

  glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MAG_FILTER, GL_NEAREST);

  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
  glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
  glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP_EXT);
  glEnable(GL_TEXTURE_CUBE_MAP_EXT);
  glEnable(GL_TEXTURE_GEN_S);
  glEnable(GL_TEXTURE_GEN_T);
  glEnable(GL_TEXTURE_GEN_R);
  glEnable(GL_NORMALIZE);
  glMatrixMode(GL_TEXTURE);
  //glScalef(1,-1,1);
  glRotatef(180,0.0,0.0,1.0);

}
```

```
//Event-driven function
void Callback(int id){
	int i;
	FILE *fp;
	switch(id){
		case APERTURE_SPIN :
			aperture=aprtr->get_float_val();
			glutPostRedisplay();
			break;
		case RADIUS_SPIN  :
			r=radius->get_float_val();
			glutPostRedisplay();
			break;
		case RESET	:
			aperture=M_PI/2.0;
			r=1.0;
			aprtr->set_float_val(aperture);
			radius->set_float_val(1.0);
			glDisable(GL_TEXTURE_CUBE_MAP_EXT);
			for(i=0;i<6;i++){
				if(image2[i]!=NULL){
					free(image2[i]->pixels);
					free(image2[i]);
					free(fileName2[i]);
					faces[i]->set_text("");
				}
			}
			selection=0;
			break;
		case EXIT	:
			for(i=0;i<6;i++){
				if(image2[i]!=NULL){
					free(image[i]);
				}

			}
			exit (EXIT_SUCCESS);
			break;
		case X	:
			if(x_axis->get_int_val() && !y_axis->get_int_val()  && !z_axis->get_int_val() ){
				rotator->enable();
				rx=1.0;
			}
			else if(x_axis->get_int_val() && (y_axis->get_int_val() || z_axis->get_int_val())){
				rx=1.0;
			}
			else if(!x_axis->get_int_val() && (y_axis->get_int_val() || z_axis>get_int_val())){
				rx=0.0;
			}
			else{
				rx=0.0;
				rotator->disable();
			}
```

```
                break;
case Y          :
                if(y_axis->get_int_val() && !x_axis->get_int_val() && !z_axis->get_int_val() ){
                        rotator->enable();
                        ry=1.0;
                }
                else if(y_axis->get_int_val() && (x_axis->get_int_val() || z_axis->get_int_val())){
                        ry=1.0;
                }
                else if(!y_axis->get_int_val() && (x_axis->get_int_val() || z_axis>get_int_val())){
                        ry=0.0;
                }
                else{
                        ry=0.0;
                        rotator->disable();
                }
                break;
case Z          :
                if(z_axis->get_int_val() && !x_axis->get_int_val() && !y_axis->get_int_val() ){
                        rotator->enable();
                        rz=1.0;
                }
                else if(z_axis->get_int_val() && (x_axis->get_int_val() || y_axis->get_int_val())){
                        rz=1.0;
                }
                else if(!z_axis->get_int_val() && (x_axis->get_int_val() || y_axis>get_int_val())){
                        rz=0.0;
                }
                else{
                        rz=0.0;
                        rotator->disable();
                }
                break;

case R          :
                glMatrixMode(GL_TEXTURE);
                glRotatef(5,rx,ry,rz);
                break;
case CCW        :
                if(z_axis->get_int_val() || x_axis->get_int_val() || y_axis->get_int_val()){
                        rx=-rx;
                        ry=-ry;
                        rz=-rz;
                }
                break;
case PX         :
                if(ext(faces[0]->get_text()) ){
                                //if(image2[0]!=NULL){
                                //      free(image2[0]);
                                //}
                        fileName2[0]=(char *)malloc(strlen(faces[0]->get_text())*sizeof(char));
                        strcpy(fileName2[0],faces[0]->get_text());
                        fp=fopen(fileName2[0], "rb");
                        image2[0]=gliReadTGA(fp,fileName2[0]);

                }
```

```
                break;
case NX         :
                if(ext(faces[1]->get_text())) {
                                //if(image2[1]!=NULL){
                                //      free(image2[1]);
                                //}
                        fileName2[1]=(char *)malloc(strlen(faces[1]->get_text())*sizeof(char));
                        strcpy(fileName2[1],faces[1]->get_text());
                        fp=fopen(fileName2[1], "rb");
                        image2[1]=gliReadTGA(fp,fileName2[1]);

                }
                break;
case PY         :
                if(ext(faces[2]->get_text()) ){
                                //if(image2[2]!=NULL){
                                //      free(image2[2]);
                                //}
                        fileName2[2]=(char *)malloc(strlen(faces[2]->get_text())*sizeof(char));
                        strcpy(fileName2[2],faces[2]->get_text());
                        fp=fopen(fileName2[2], "rb");
                        image2[2]=gliReadTGA(fp,fileName2[2]);

                }
                break;
case NY         :
                if(ext(faces[3]->get_text()) ){
                                //if(image2[3]!=NULL){
                                //      free(image2[3]);
                                //}
                        fileName2[3]=(char *)malloc(strlen(faces[3]->get_text())*sizeof(char));
                        strcpy(fileName2[3],faces[3]->get_text());
                        fp=fopen(fileName2[3], "rb");
                        image2[3]=gliReadTGA(fp,fileName2[3]);

                }
                break;
case PZ         :
                if(ext(faces[4]->get_text()) ){
                                //if(image2[4]!=NULL){
                                //      free(image2[4]);
                                //}
                        fileName2[4]=(char *)malloc(strlen(faces[4]->get_text())*sizeof(char));
                        strcpy(fileName2[4],faces[4]->get_text());
                        fp=fopen(fileName2[4], "rb");
                        image2[4]=gliReadTGA(fp,fileName2[4]);

                }
                break;
case NZ         :
                if(ext(faces[5]->get_text())){
                                //if(image2[5]!=NULL){
                                //      free(image2[5]);
                                //}
                        fileName2[5]=(char *)malloc(strlen(faces[5]->get_text())*sizeof(char));
                        strcpy(fileName2[5],faces[5]->get_text());
```

```
                                        fp=fopen(fileName2[5], "rb");
                                        image2[5]=gliReadTGA(fp,fileName2[5]);

                                    }
                                    break;
                case NEW            :
                                    if(image2[0]!=NULL && image2[1]!=NULL && image2[2]!=NULL &&
image2[3]!=NULL && image2[4]!=NULL && image2[5]!=NULL && selection==0){
                                        glDisable(GL_TEXTURE_CUBE_MAP_EXT);
                                        Setup(image2);
                                        selection=1;
                                        //glMatrixMode(GL_TEXTURE);
                                        //glScalef(1,-1,1);
                                        //glRotatef(180,0.0,0.0,1.0);
                                    }
                                    break;
        }

}


static void
display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  DrawSphere();

  //glMatrixMode(GL_TEXTURE);
  //glScalef(-1,1,-1);
                //glMatrixMode(GL_MODELVIEW);
  glutSwapBuffers();


}

//Key function
void
keyFunc (unsigned char key, int x, int y)
{
        int i;
   switch (key)
   {
      case '\033':
            for(i=0;i<6;i++){
                //free(image[i]->pixels);
                free(image[i]);
             }
         exit (EXIT_SUCCESS);
         break;


         case 'i':
```

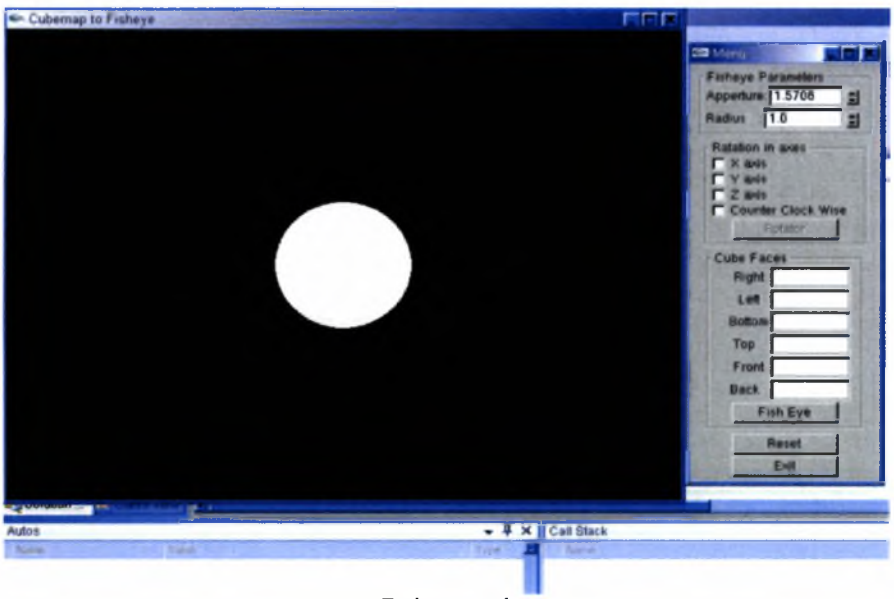
```
                        r+=0.01f;
                        radius->set_float_val(r);

                        glutPostRedisplay();
                        //printf("r:%f\n",r);


          break;
        case 'd':


                        r-=0.01f;
                        radius->set_float_val(r);

                        glutPostRedisplay();


                        //printf("r:%f\n",r);

          break;

        case 'x':
                        glMatrixMode(GL_TEXTURE);
                        glRotatef(5,1.0,0.0,0.0);

                        glutPostRedisplay();

          break;

case 'X':

                //}
                        glMatrixMode(GL_TEXTURE);
                        glRotatef(-5,1.0,0.0,0.0);

                        glutPostRedisplay();

          break;

        case 'a':


                        aperture-=0.01;
                        aprtr->set_float_val(aperture);
                        glutPostRedisplay();

                        //printf("aperture:%f\n",aperture);

                break;
        case 'A':


                        aperture+=0.01;

                        aprtr->set_float_val(aperture);
```

```
                        glutPostRedisplay();

                        //printf("aperture:%f\n",aperture);

                break;

        case 'y':

                        glMatrixMode(GL_TEXTURE);
                        glRotatef(5,0.0,1.0,0.0);

                        glutPostRedisplay();

          break;

    case 'Y':

                        glMatrixMode(GL_TEXTURE);
                        glRotatef(-5,0.0,1.0,0.0);

                        glutPostRedisplay();

          break;

        case 'z':


                        glMatrixMode(GL_TEXTURE);
                        glRotatef(5,0.0,0.0,1.0);

                        glutPostRedisplay();

          break;

    case 'Z':


                        glMatrixMode(GL_TEXTURE);
                        glRotatef(-5,0.0,0.0,1.0);

                        glutPostRedisplay();

          break;
        }
        //glutSwapBuffers();


}

//Window-resize- driven function
void
reshapeFuncZoom (int w, int h)
{
                GLfloat fAspect;
```

```
                if(h == 0)
                        h = 1;


                glViewport(0, 0, w, h);

                fAspect = (GLfloat)w/(GLfloat)h;

                // Reset coordinate system
                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();

                // Produce the perspective projection
                gluPerspective(360*atan2(1.5,2*ZTRANS -5.6)/M_PI, fAspect, 2*ZTRANS -5.6, 2*ZTRANS + 2.0);

                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();
                glutPostRedisplay();

}

void glutIdle(){
        glutSetWindow(main_window);
        glutPostRedisplay();
}

//Set Elements for glui menu
void gl(){
  glutDisplayFunc(display);
  GLUI_Master.set_glutSpecialFunc( NULL );
  GLUI_Master.set_glutIdleFunc(glutIdle);
  GLUI_Master.set_glutMouseFunc(NULL);
  GLUI_Master.set_glutKeyboardFunc(keyFunc);
  GLUI_Master.set_glutReshapeFunc(reshapeFuncZoom);

  //glui_subwin= GLUI_Master.create_glui_subwindow(main_window,GLUI_SUBWINDOW_RIGHT);
  glui_subwin=GLUI_Master.create_glui("Menu",0,0,0);
  glui_subwin->set_main_gfx_window(main_window);
  fisheye_parameters=glui_subwin->add_panel("Fisheye Parameters",GLUI_PANEL_EMBOSSED);
  rotation=glui_subwin->add_panel("Ratation in axes",GLUI_PANEL_EMBOSSED);
  cube_faces=glui_subwin->add_panel("Cube Faces",GLUI_PANEL_EMBOSSED);
  aprtr=glui_subwin-
>add_spinner_to_panel(fisheye_parameters,"Apperture:",GLUI_SPINNER_FLOAT,NULL,APERTURE_SPIN,Callback)
;
  aprtr->set_float_limits(0.21,3.14159,GLUI_LIMIT_CLAMP);
  aprtr->set_speed(1.0);
  aprtr->set_float_val(aperture);

  radius=glui_subwin->add_spinner_to_panel(fisheye_parameters,"Radius
:",GLUI_SPINNER_FLOAT,NULL,RADIUS_SPIN,Callback);
  radius->set_float_limits(0.21,3.0,GLUI_LIMIT_CLAMP);
  radius->set_speed(1.0);
  radius->set_float_val(r);
  x_axis=glui_subwin->add_checkbox_to_panel(rotation,"X axis", NULL, X, Callback);
```

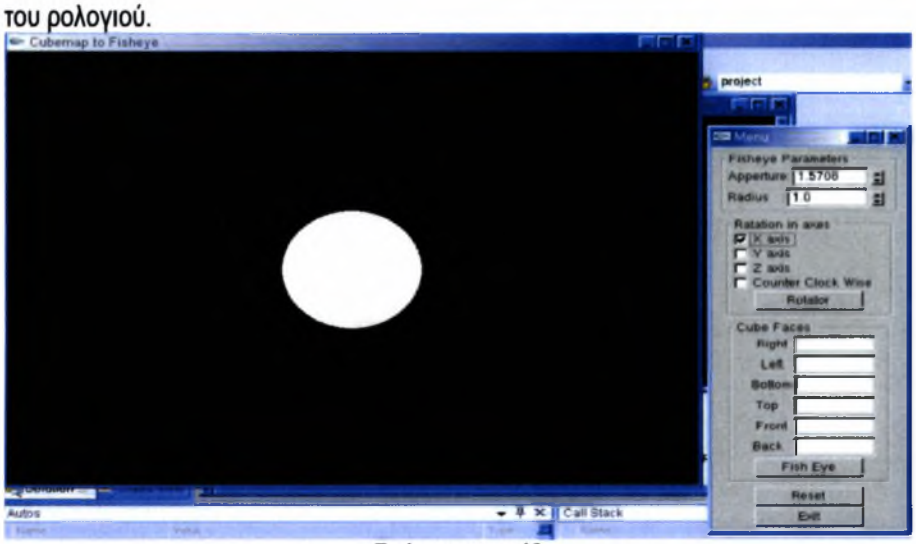
```
 y_axis=glui_subwin->add_checkbox_to_panel(rotation,"Y axis", NULL, Y, Callback);
 z_axis=glui_subwin->add_checkbox_to_panel(rotation,"Z axis", NULL, Z, Callback);
 count_clock=glui_subwin->add_checkbox_to_panel(rotation,"Counter Clock Wise", NULL, CCW, Callback);
 reset=glui_subwin->add_button("Reset",RESET,Callback);
 end=glui_subwin->add_button("Exit",EXIT,Callback);
 rotator=glui_subwin->add_button_to_panel(rotation,"Rotator",R,Callback);
 rotator->disable();
 faces[0]=glui_subwin->add_edittext_to_panel(cube_faces,"Right ",GLUI_EDITTEXT_TEXT,NULL,PX,Callback);
 faces[1]=glui_subwin->add_edittext_to_panel(cube_faces,"Left  ",GLUI_EDITTEXT_TEXT,NULL,NX,Callback);
 faces[2]=glui_subwin->add_edittext_to_panel(cube_faces,"Bottom",GLUI_EDITTEXT_TEXT,NULL,PY,Callback);
 faces[3]=glui_subwin->add_edittext_to_panel(cube_faces,"Top   ",GLUI_EDITTEXT_TEXT,NULL,NY,Callback);
 faces[4]=glui_subwin->add_edittext_to_panel(cube_faces,"Front ",GLUI_EDITTEXT_TEXT,NULL,PZ,Callback);
 faces[5]=glui_subwin->add_edittext_to_panel(cube_faces,"Back  ",GLUI_EDITTEXT_TEXT,NULL,NZ,Callback);
 new_Fish_Eye=glui_subwin->add_button_to_panel(cube_faces,"Fish Eye",NEW,Callback);
}


int
main(int argc, char **argv)
{
 int i;



 for(i=0;i<6;i++){
          image2[i]=NULL;
 }



 glutInitWindowSize(640, 480);
 glutInit(&argc, argv);

 glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH | GLUT_RGBA);
 main_window=glutCreateWindow("Cubemap to Fisheye");

 /* Run-time extension check. */
 if (!glutExtensionSupported("GL_EXT_texture_cube_map")) {
  fprintf(stderr,
    "cubemap: Requires the EXT_texture_cube_map OpenGL extension to run.\n");
  exit(0);
 }

gl();
 //Setup(image);
 glutMainLoop();
 return 0;
}
```

Στην ουσία φτιάχνεται ένα παράθυρο 640x480 (*Εικόνα Αρχική*) όπως φαίνεται στην main( ). Σε αυτό το παράθυρο αρχικά εμφανίζεται μια λευκή σφαίρα η οποία φτιάχνεται από την κλήση της συνάρτησης DrawShpere( ) εντός της συνάρτησης display( ), στη συνάρτηση gl( ). Επίσης, στην συνάρτηση gl( ) αρχικοποιούνται και τα στοιχεία του μενού glui.



Εικόνα αρχική

Τα στοιχεία του μενού που μπορούν να χρησιμοποιηθούν για την πρόκληση γεγονότων όπως τα κουμπιά και τα radio-buttons συνδέονται με την συνάρτηση Callback(int id), η οποία πυροδοτεί νέες συμπεριφορές ανάλογα με το εκάστοτε γεγονός. Στο ανώτερο υπομενού έχουμε κάποια στοιχεία για το fisheye, το aperture και την ακτίνα της σφαίρας(Radius). Στο δεύτερο (*Εικόνα υπομενού 2*)έχουμε κάποια radio-buttons των οποίων η επιλογή μπορεί να επιτρέπει την περιστροφή ως προς έναν ή περισσότερους άξονες, κατά την φορά του ρολογιού ή αντίθετα της φοράς του ρολογιού.



Εικόνα υπομενού2

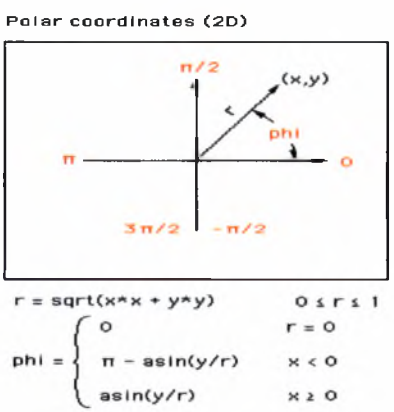
Τέλος η περιστροφή επιτυγχάνεται επιλέγοντας το κουμπί Rotator, το οποίο ενεργοποιείται μόνο αν τουλάχιστον ένα από τα radio-buttons που είναι υπεύθυνα για τους άξονες είναι ενεργοποιημένο.

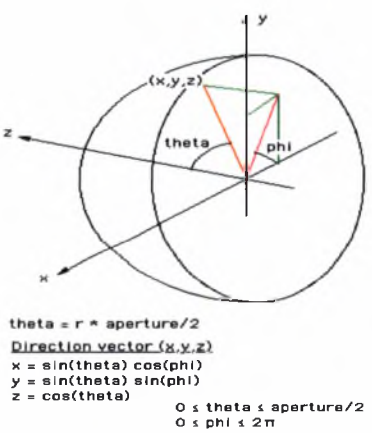
Τέλος, στο υπομενού Cubemap Faces, υπάρχουν 6 editText όπου γράφουμε τα ονόματα των εικόνων tga μαζί με την κατάληξή τους. Όταν γραφτεί το όνομα της εικόνας, τότε μέσω της

συνάρτησης Callback(int id), διαβάζεται η εικόνα. Όταν έχουν συμπληρωθεί τα 6 πεδία του υπομενού, και εφόσον πρόκειται για εικόνες τετραγωνικές ίδιων διαστάσεων, τότε πατώντας το κουμπί Fish Eye, δημιουργείται ένα angular Fisheye (εικόνα *Fisheye*)από τις 6 εικόνες του Cube map μας.

FishEye

Η δημιουργία του angular Fisheye γίνεται με την κλήση της συνάρτησης Setup(gliGenericImage *im[]) από την Callback(int id). Εκεί γίνεται το texture mapping των 6 εικόνων με την χρήση του cubemap extension. Επειδή όμως θέλουμε να δημιουργήσουμε angular Fisheye, όταν ζωγραφίζουμε την σφαίρα με την συνάρτηση DrawShpere(), καλούμε αντί της glVertex, καλούμε την Vert(float *theta, float *phi), όπου ζωγραφίζουμε ανά τετράδες σημείων και για κάθε σημείο ορίζουμε το κατάλληλο διάνυσμα διεύθυνσης προκειμένου να έχουμε την κατάλληλη προβολή πάνω στην σφαίρα.





Να σημειωθεί ότι πολλές λειτουργίες που εκτελούνται από το glui μενού (αύξηση/μείωση ακτίνας/aperture, περιστροφή προς άξονα x,y ή z,τερματισμός προγράμματος)εκτελούνται και μέσω πληκτρολογίου χάρη στην συνάρτηση void keyFunc( )(όταν είναι επιλεγμένο το παράθυρο όπου πραγματοποιείται το rendering).

5 Σύνοψη

Αυτό που είδατε είναι η μετατροπή ενός cube map σε angular fisheye. Το πρόγραμμα δοκιμάστηκε σε υπολογιστή AMD Athlon 64 3200 με κάρτα Ram 1GB και κάρτα γραφικών GeForce 6600 GT. Το format εικόνας που χρησιμοποιείται είναι tga. Ο παραπάνω κώδικας δύναται να χρησιμοποιηθεί τόσο σε Winows όσο και σε Linux Ο λόγος της χρήσης του angular fisheye είναι ότι είναι πολύ ευκολότερο στην δημιουργία του και επιπλέον είναι πολύ πιο εύχρηστο από άλλα είδη fisheye προβολών, μια και οι παραμορφώσεις της εικόνας τόσο στην περιφέρεια όσο και προς στο κέντρο είναι ίδιες. Μάλιστα με την χρήση του cube map αποφεύγουμε να χρησιμοποιήσουμε το φτωχό σε ποιότητα sphere mapping ή το χρονοβόρο dual paraboloid mapping, οπότε κερδίζουμε και σε ποιότητα και σε ταχύτητα. Αυτό το πρόγραμμα θα μπορούσε να επεκταθεί περεταίρω, όπως για παράδειγμα να χρησιμοποιεί και άλλα format εικόνων, πέρα από το tga, όπως γίνεται εδώ για παράδειγμα, καθώς και να τις αποθηκεύει στο ίδιο format ή σε άλλο. Επιπλέον θα ήταν μπορούσαν να προστεθούν επιλογές και για άλλες παραλλαγές προβολής fish eye. Τέλος το πρόγραμμα θα μπορούσε να γίνει ταχύτερο χρησιμοποιώντας είτε τεχνικές ταυτόχρονου προγραμματισμού είτε άλλες βιβλιοθήκες.

Βιβλιογραφία

Παρατίθενται οι εξής τίτλοι βιβλίων και ιστοσελίδες:

- Transformations and Projections in Computer Graphics του David Salomon
- OpenGL SuperBible 4th edition των Richard S. Wright, Benjamin Lipchak και Nicholas Haemel
- http://developer.nvidia.com/object/cube_map_ogl_tutorial.html
- http://local.wasp.uwa.edu.au/~pbourke/projection/fisheye/
- http://local.wasp.uwa.edu.au/~pbourke/projection/cube2dome/
- http://local.wasp.uwa.edu.au/~pbourke/projection/skyvision/
- http://local.wasp.uwa.edu.au/~pbourke/projection/dometexture/
- http://artis.imag.fr/Publications/2008/GHFP08/i3d2007.pdf
- http://perception.inrialpes.fr/Publications/2007/TSR07/TardifSturmRoy-iccv2007.pdf
- http://www.cg.tuwien.ac.at/courses/CG23/software/sig99advopengl.PDF