Γραφική με Υπολογιστές - Θέαση

Γκούντρας Ιωάννης

AEM:10332

8 Ιουλίου, 2024

Abstract

Σε αυτή την εργασία χρησιμοποιώντας τους αλγορίθμους που υλοποιήθηκαν στις προηγούμενες, υλοποιείται ένα πλήρες πλαίσιο δημιουργίας φωτογραφιών μιας εικονικής σκηνής, εφαρμόζοντας το πλήρες μοντέλο φωτισμού.

1 Φωτισμός και Υλικό επιφάνειας

Υλοποιήθηκε η συνάρτηση `light()` η οποία χρησιμοποιεί το πλήρες μοντέλο φωτισμού που διδάχθηκε στο μάθημα και έχοντας ως είσοδο ένα σημείο, το normal vector και το color vector που του αντιστοιχούν καθώς και τις θέσεις και εντάσεις των πηγών και τους συντελεστές ανακλάσεων, επιστρέφει το τελικό χρώμα που αντιστοιχεί στο σημείο.

Να σημειωθεί πως ενώ δεν ήταν ζητούμενο από την εκφώνηση κρίθηκε απαραίτητο να συμπεριληφθεί στα ορίσματα της συνάρτησης και το διάνυσμα της έντασης της διάχυτης ακτινοβολίας του περιβάλλοντος.

Η συνάρτηση υπολογίζει αρχικά το διάνυσμα V που εξαρτάται μόνο από τη θέση της κάμερας και του δοθέντος σημείου και στη συνέχεια αρχικοποιεί ένα μηδενικό διάνυσμα I. Αφού υπολογίσει τη συνιστώσα που οφείλεται στον ambient φωτισμό και την προσθέσει στο I, στη συνέχεια για κάθε μία από τις πηγές υπολογίζει το διάνυσμα L και αφού υπολογίσει τις συνιστώσες που οφείλονται στο diffuse και στο specular φωτισμό τις προσθέτει στο συνολικό I.

Τέλος κάνει clip το I ανάμεσα στις τιμές 0 και 1, για να βεβαιωθεί ότι δεν θα υπάρξει κάποιο σφάλμα στη συνέχεια όταν το χρώμα θα πολλαπλασιαστεί με 255, και επιστρέφει την τιμή του I.

2 Υπολογισμός κανονικών διανυσμάτων επιφάνειας

Είναι απαραίτητο να υπολογίζεται για κάθε ένα από τα τρίγωνα του αντικειμένου το normal vector του ώστε να μπορέσει να χρησιμοποιηθεί το πλήρες μοντέλο φωτισμού. Για αυτόν τον λόγο υλοποιήθηκε η συνάρτηση `calculate_normals()` η οποία έχοντας ως είσοδο το σύνολο των κορυφών και το σύνολο των τριγώνων (faces) επιστρέφει το σύνολο των normals για κάθε vertex.

Αρχικοποιεί έναν πίνακα normals ίδιου μεγέθους με αυτό των σημείων και στη συνέχεια για κάθε face χρησιμοποιώντας τα vertices V_1, V_2 και V_3 που το αποτελούν, υπολογίζει τα διανύσματα $\text{Vec}_1 = V_2 - V_1$ και $\text{Vec}_2 = V_3 - V_1$. Στη συνέχεια υπολογίζεται το εξωτερικό γινόμενο αυτών των δύο και κανονικοποιείται διαιρώντας με το μέτρο του. Τέλος χρησιμοποιώντας σαν δείκτη στον πίνακα vectors τους δείκτες που χρησιμοποιούνται και την προσπέλαση των vertices, ορίζονται τα normals στο σωστό σημείο. Επιστρέφεται ο πίνακας normals που περιέχει τα normal vectors που αντιστοιχούν στα vertices, με την ίδια σειρά.

3 Συνάρτηση φωτογράφισης

Σε παρόμοια λογική με την `render_image()` του προηγούμενου παραδοτέου υλοποιείται η `render_object()` που είναι υπεύθυνη για την φωτογράφιση του αντικειμένου.

Αρχικά δημιουργεί έναν M επί N πίνακα img που κάθε στοιχείο του είναι το `bg_color` διάνυσμα, υπολογίζει όλα τα normal vectors και στη συνέχεια ακολουθεί τη διαδικασία της προηγούμενης εργασίας βρίσκοντας τις συντεταγμένες των προβεβλημένων σημείων καθώς και των pixels μέσω rasterisation.

Στη συνέχεια αρχικοποιείται ένας πίνακας `triangles` στον οποίο γίνονται append dictionaries `triangle` που περιέχουν για κάθε τρίγωνο τις εξής πληροφορίες:

- vertices
- normals
- color
- depth
- points_2d
- bcoords

Τα `bcoords` υπολογίζονται προσθέτοντάς τα διανύσματα των κορυφών του τριγώνου και διαιρώντας με τον αριθμό τους.

Έξτρα παραδοτέο

Για το έξτρα παραδοτέο ακολουθεί μια δεύτερη for loop που χρησιμοποιεί τα `face_uv_indices` για να προσπελάσει τα διανύσματα των uv συντεταγμένων που αντιστοιχούν σε κάθε κορυφή του τριγώνου και στη συνέχεια προσθέτει σε κάθε dictionary του πίνακα `triangles` ένα νέο key ”uvs” που περιέχει έναν πίνακα με τα τρία uv διανύσματα του τριγώνου.

Ο πίνακας `triangles` στη συνέχεια γίνεται reverse sort με βάση το βάθος του κάθε τριγώνου ώστε να ξεκινάει με τα πιο απομακρυσμένα τρίγωνα που θέλουμε να πληρωθούν πρώτα.

Τέλος ανάλογα με το όρισμα shader της συνάρτησης καλείται η `shade_gouraud` ή η `shade_phong` για κάθε ένα από τα τρίγωνα.

4 Shade Gouraud

Αυτή η συνάρτηση αρχικά υπολογίζει για κάθε χρώμα του πίνακα εισόδου `vertsc` το τελικό χρώμα μέσω της συνάρτησης `light` και τα αποθηκεύει σε έναν νέο πίνακα `colors`. Στη συνέχεια καλεί την συνάρτηση `g_shading` του πρώτου παραδοτέου η οποία επιστρέφει την ανανεωμένη εικόνα με βαμμένο το τρίγωνο, που με τη σειρά της επιστρέφεται στην `render_object`. Έτσι μέσω της επανάληψης στην `render_object` βάφονται όλα τα τρίγωνα.

Έξτρα παραδοτέο

Για το έξτρα παραδοτέο ο πίνακας εισόδου `vertsc` αγνοείται. Η συνάρτηση χρησιμοποιεί σαν έξτρα ορίσματα εισόδου τα uv διανύσματα του τριγώνου και τον πίνακα που αντιστοιχεί στην φωτογραφία που χρησιμοποιείται ως texture map, και καλώντας τη συνάρτηση `bilerp` για κάθε ένα από τα τρία uv διανύσματα υπολογίζει το χρώμα που αντιστοιχεί στο κάθε vertex. Αυτά τα χρώματα στη συνέχεια αποθηκεύονται στον πίνακα `colors` και δίνονται σαν όρισμα στην `g_shading`, η οποία δεν αλλάζει.

5 Shade Phong

Η συγκεκριμένη συνάρτηση είναι σε πολύ μεγάλο βαθμό βασισμένη στην συνάρτηση `g_shading` του πρώτου παραδοτέου, με κάποιες επιπλέον προσθήκες.

Αρχικά σε κάθε ένα από τα dictionaries που αντιπροσωπεύουν τα edges του τριγώνου, προστίθεται η πληροφορία ”top_normal” και ”bottom_normal” που είναι τα normal vectors του vertex που βρίσκεται ψηλότερα και χαμηλότερα για δεδομένο edge.

Έξτρα παραδοτέο

Για το έξτρα παραδοτέο προστίθεται επίσης με την ίδια λογική σε κάθε dictionary των edges και η πληροφορία ”top_uv” και ”bottom_uv” που είναι τα uv vectors του vertex που βρίσκεται ψηλότερα και χαμηλότερα για δεδομένο edge.

Στη συνέχεια αφού βρεθούν τα active edges για κάθε ένα από αυτά εκτός από το interpolation που γινόταν για το χρώμα γίνεται ακριβώς με την ίδια λογική interpolate και ανάμεσα στα normal vectors. Αυτό γίνεται ανάμεσα στα top και bottom normal vectors του edge για τη δεδομένη τιμή του y.

Έξτρα παραδοτέο

Για το έξτρα παραδοτέο γίνεται επίσης με την ίδια λογική interpolation ανάμεσα στα top και bottom uv vectors του edge για τη δεδομένη τιμή του y.

Έπειτα, στον υπολογισμό των active points όπου μέχρι στιγμής αποθηκευόταν για κάθε active point ένας πίνακας που περιείχε τις συντεταγμένες x και y του καθώς και το χρώμα του, τώρα προστίθεται και το αντίστοιχο normal vector που βρέθηκε από το interpolation.

Έξτρα παραδοτέο

Για το έξτρα παραδοτέο προστίθεται επίσης στον πίνακα αυτό και το αντίστοιχο uv vector που βρέθηκε από το interpolation.

Αφού γίνουν sort τα active points από το αριστερότερο στο δεξιότερο, παλαιότερα γινόταν interpolate το χρώμα του κάθε σημείου του scanline και πλέον γίνεται επιπλέον interpolate για το normal vector του κάθε σημείου, ανάμεσα στα normal vectors των active points με βάση το δεδομένο x.

Τέλος καλείται η συνάρτηση `light` στην οποία δίνονται ως ορίσματα το normal vector καθώς και το χρώμα του σημείου και επιστρέφεται το τελικό χρώμα του σημείου, με το οποίο βάφεται το αντίστοιχο pixel της φωτογραφίας.

Έξτρα παραδοτέο

Για το έξτρα παραδοτέο αγνοείται το interpolated χρώμα του σημείου. Αντί για αυτό, γίνεται interpolate ανάμεσα στα uv vectors των active points για δεδομένο x και χρησιμοποιείται η συνάρτηση `bilerp` για να υπολογιστεί το χρώμα του σημείου μέσω του texture map. Στη συνέχεια αυτό το χρώμα δίνεται σαν όρισμα στη συνάρτηση `light` με την ίδια λογική όπως προηγουμένως, η οποία επιστρέφει το τελικό χρώμα με το οποίο βάφεται το αντίστοιχο pixel της εικόνας.

6 Συνάρτηση bilerp - Έξτρα Παραδοτέο

Μέχρι στιγμής αναλύθηκε το πως εμπλουτίστηκαν οι συναρτήσεις ώστε να δουλεύουν χρησιμοποιώντας το texture map καθώς και τα συγκεκριμένα σημεία στα οποία έπρεπε να γίνουν αλλαγές. Γενικά δεν προτιμήθηκε να διαγραφούν εντελώς τα κομμάτια του κώδικα που χρησιμοποιούν τα χρώματα από το αρχείο `hw3.npy`, αλλά απλώς να αγνοηθούν, για να μπορεί να γίνεται testing και με τις δύο μεθόδους πιο εύκολα. Αυτό ίσως προσθέτει παραπάνω καθυστέρηση σε κάθε render αλλά δεν αποτέλεσε πρόβλημα.

Η συνάρτηση `bilerp` δέχεται σαν ορίσματα εισόδου τις uv συντεταγμένες και το texture map. Αρχικά υπολογίζει τα ακριβή σημεία x και y που αντιστοιχούν στην εικόνα, και στη συνέχεια ορίζει ως x_1 και y_1 τους αμέσως προηγούμενους ακεραίους και x_2 και y_2 τους αμέσως επόμενους.

Διαβάζει στη συνέχεια από το texture map τα διανύσματα των χρωμάτων που αντιστοιχούν στις τέσσερις κορυφές $(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)$ και υπολογίζει τα dx και dy. Όπως φαίνεται και στην φωτογραφία 1, για το δοθέν πορτοκαλί σημείο, γίνεται πρώτα interpolation ανάμεσα στα αριστερά και δεξία σημεία με βάση το dy και προκύπτουν τα color vertices που θα αντιστοιχούσαν στα κόκκινα σημεία και τέλος γίνεται ένα interpolation ανάμεσα στα κόκκινα σημεία με βάση το dx και προκύπτει το χρώμα που αντιστοιχεί στο πορτοκαλί σημείο.

Τέλος το τελικό χρώμα γίνεται clip ανάμεσα στο 0 και το 1 για να μην υπάρξουν προβλήματα μετέπειτα. Αξίζει να σημειωθεί εδώ ότι όταν φορτώνεται το texture map σαν πίνακας από την φωτογραφία που μας δίνεται, τα χρωματικά διανύσματα διαιρούνται με το 255 ώστε να συμβαδίζουν με τη λογική όλων των υπολοίπων συναρτήσεων που περιμένουν χρωματικές συνιστώσες ανάμεσα στο 0 και το 1.

7 Demo

Για το demo αρχικά φορτώνεται το αρχείο `hw3.npy` ως dictionary και η `cat_diff.png` μέσω της opencv ως πίνακας. Το texture map αυτό μετατρέπεται από BGR μορφή (η οποία είναι η

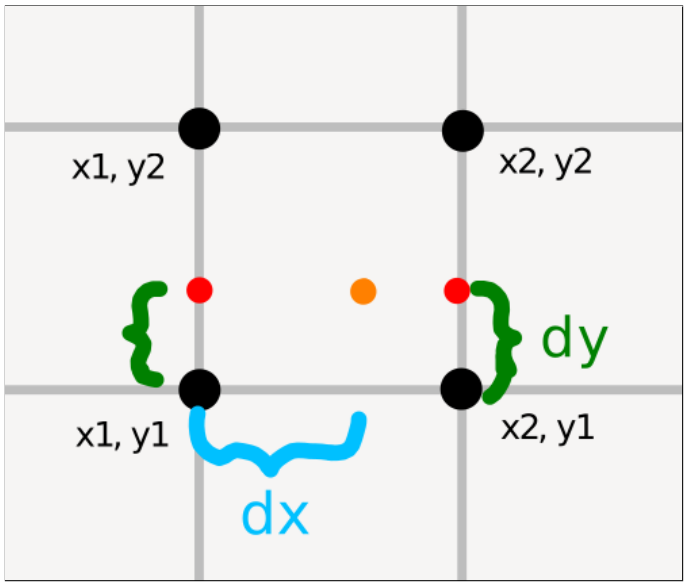

Figure 1: Ανάλυση λειτουργίας της bilerp

default της opencv) σε RGB και διαιρείται με το 255 για να περιέχει χρωματικά διανύσματα με τιμές ανάμεσα στο 0 και το 1. Στη συνέχεια καλείται η `render_object` για κάθε μία από τις φωτογραφίες που ζητείται. Πριν και μετά από κάθε κλήση της, χρησιμοποιείται η time για να υπολογίζεται ο συνολικός χρόνος που απαιτείται για το rendering. Αυτός ο χρόνος εκτυπώνεται.

Για να πετύχουμε τις φωτογραφίες που ζητούνται δίνονται σαν ορίσματά στην render τα εξής:

- Για shading = ”gouraud”:
  - Για μόνο ambient lighting k_d και k_s 0
  - Για μόνο diffuse lighting k_a και k_s 0
  - Για μόνο specular lighting k_a και k_d 0
  - Για όλους τους φωτισμούς όλα κανονικά από τα data
  - Για μόνο την πρώτη πηγή δίνεται σαν πίνακας θέσεων πηγών φωτός και εντάσεων ένας που περιέχει μόνο της πρώτης: [data[”light_positions”][0]] και [data[”light_intensities”][0]]
  - Αντίστοιχα για τις άλλες δύο πηγές χρησιμοποιώντας δείκτη 1 και 2.
- Για shading = ”phong”: Ακριβώς τα ίδια ορίσματα με την ”gouraud”

Για να αποφευχθεί η επανάληψη ίδιου κομματιού κώδικα υλοποιήθηκε η συνάρτηση `prepareImgForOpenCV()` η οποία έχοντας σαν όρισμα ένα numpy array που αντιπροσωπεύει εικόνα, το πολλαπλασιάζει με 255, το μετατρέπει από BGR σε RGB και το κάνει flip vertically (γιατί στην opencv το (0,0) αντιστοιχεί στην επάνω αριστερά γωνία, οπότε τα y αυξάνουν ”ανάποδα” με αποτέλεσμα να εμφανίζεται ανάποδα το αντικείμενο μας).

Κάθε μία από τις παραγόμενες φωτογραφίες από την render περνάει μέσα από αυτή τη συνάρτηση και στη συνέχεια αποθηκέυεται με το σωστό όνομα σε έναν φάκελο `Results` καθώς και εμφανίζεται με την `imshow` της opencv. Συνολικά παράγονται και αποθηκεύονται 14 φωτογραφίες οι οποίες ακολουθούν. Να σημειωθεί ότι αποθηκεύονται και εμφανίζονται όλες μαζί στο τέλος.

8 Αποτελέσματα Gouraud Shading με Texture Map

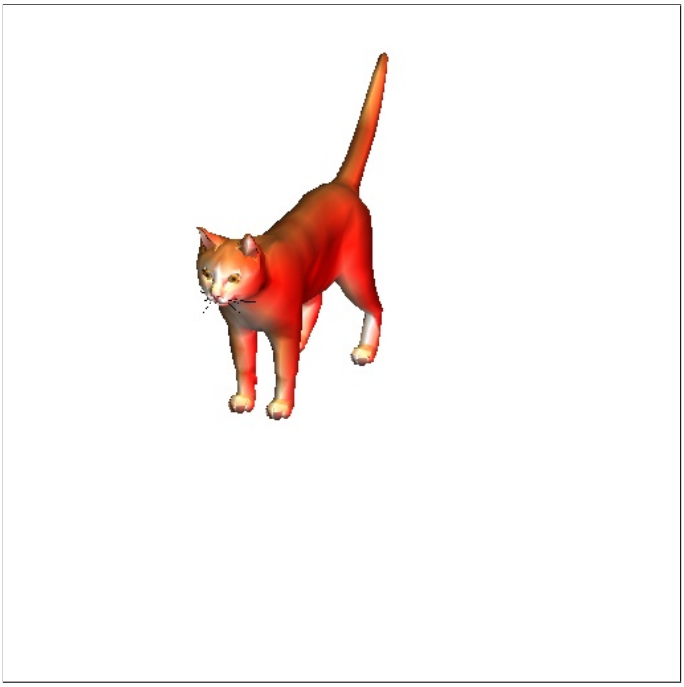

Figure 2: Gouraud Shading με όλες τις πηγές και όλα τα είδη φωτισμού

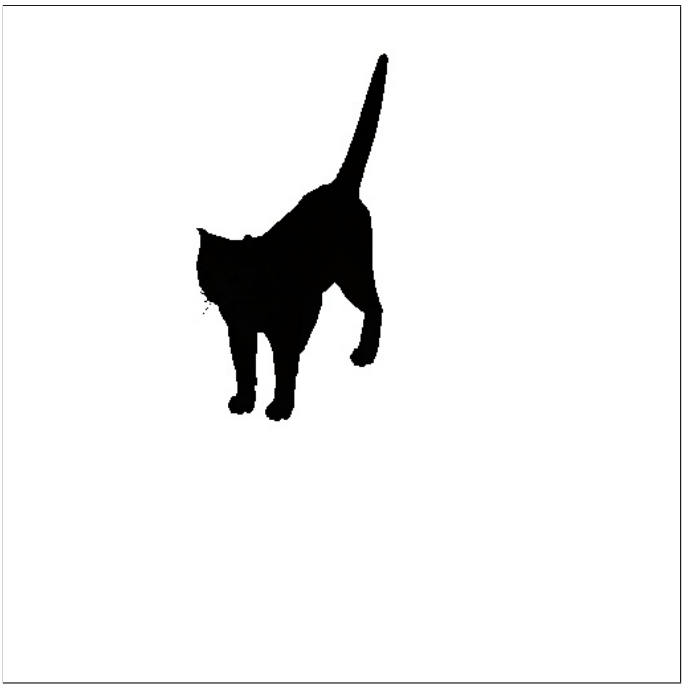

Figure 3: Gouraud Shading με όλες τις πηγές και μόνο ambient lighting

Figure 4: Gouraud Shading με όλες τις πηγές και μόνο diffuse lighting

Figure 5: Gouraud Shading με όλες τις πηγές και μόνο specular lighting

Figure 6: Gouraud Shading με μόνο την πρώτη πηγή και όλα τα είδη φωτισμού

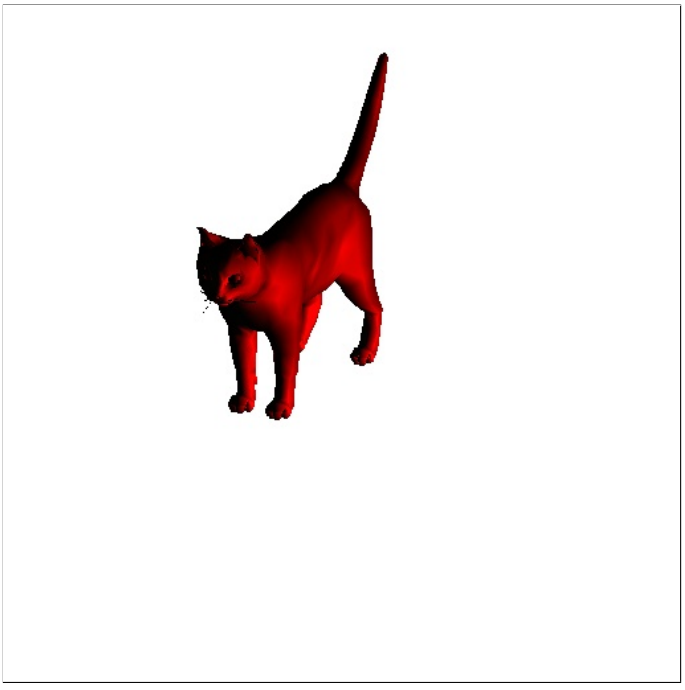

Figure 7: Gouraud Shading με μόνο την δεύτερη πηγή και όλα τα είδη φωτισμού

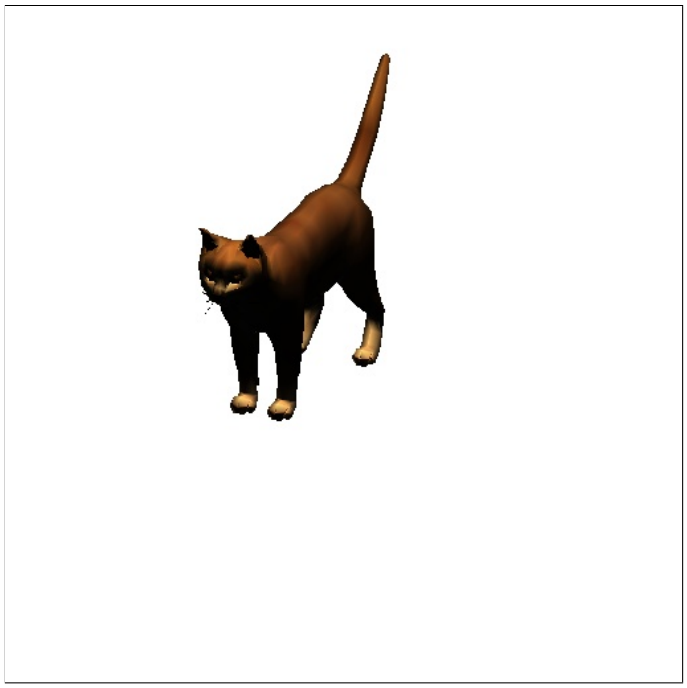

Figure 8: Gouraud Shading με μόνο την τρίτη πηγή και όλα τα είδη φωτισμού

9 Αποτελέσματα Phong Shading με Texture Map

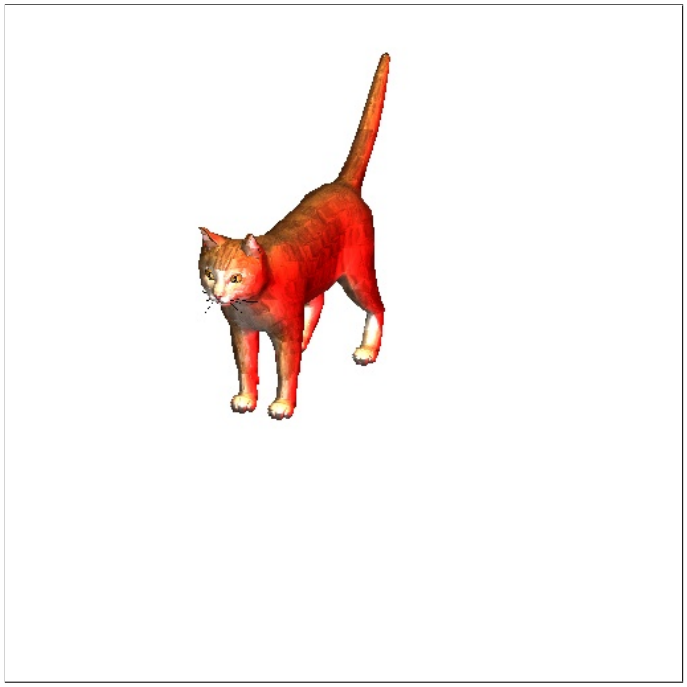

Figure 9: Phong Shading με όλες τις πηγές και όλα τα είδη φωτισμού

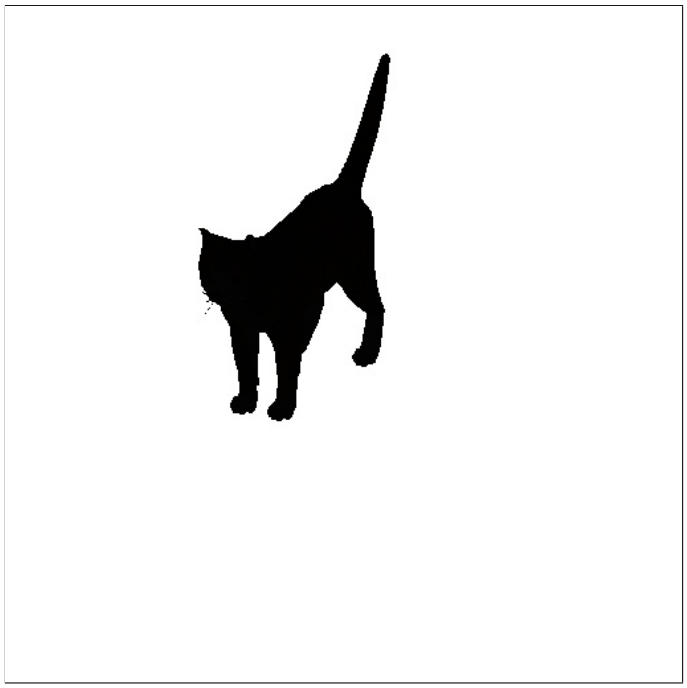

Figure 10: Phong Shading με όλες τις πηγές και μόνο ambient lighting

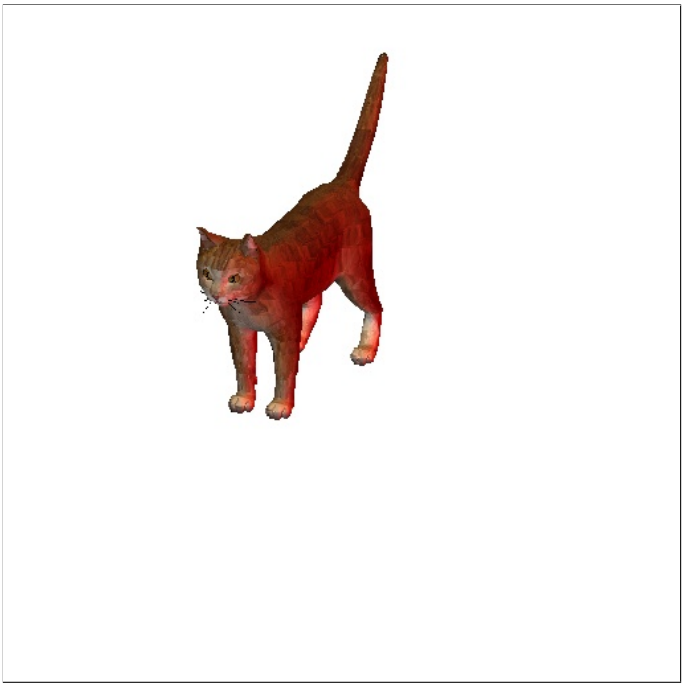

Figure 11: Phong Shading με όλες τις πηγές και μόνο diffuse lighting

Figure 12: Phong Shading με όλες τις πηγές και μόνο specular lighting

Figure 13: Phong Shading με μόνο την πρώτη πηγή και όλα τα είδη φωτισμού

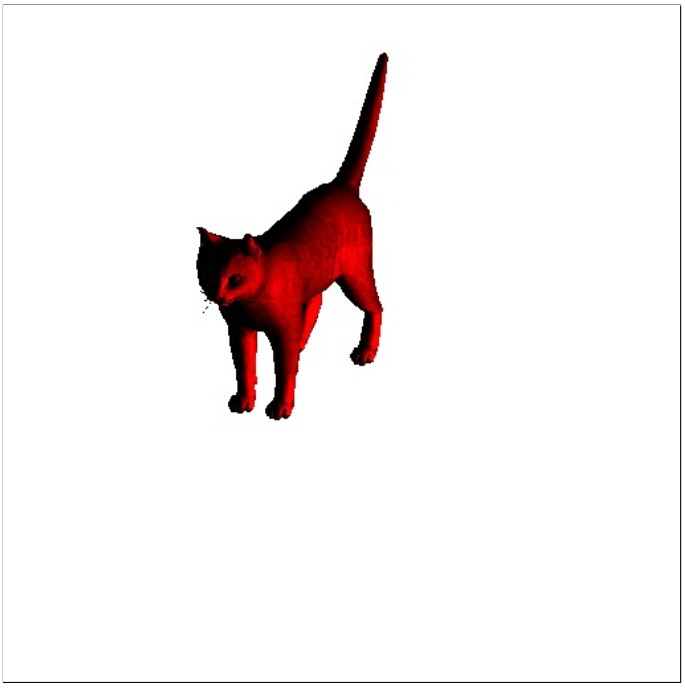

Figure 14: Phong Shading με μόνο την δεύτερη πηγή και όλα τα είδη φωτισμού

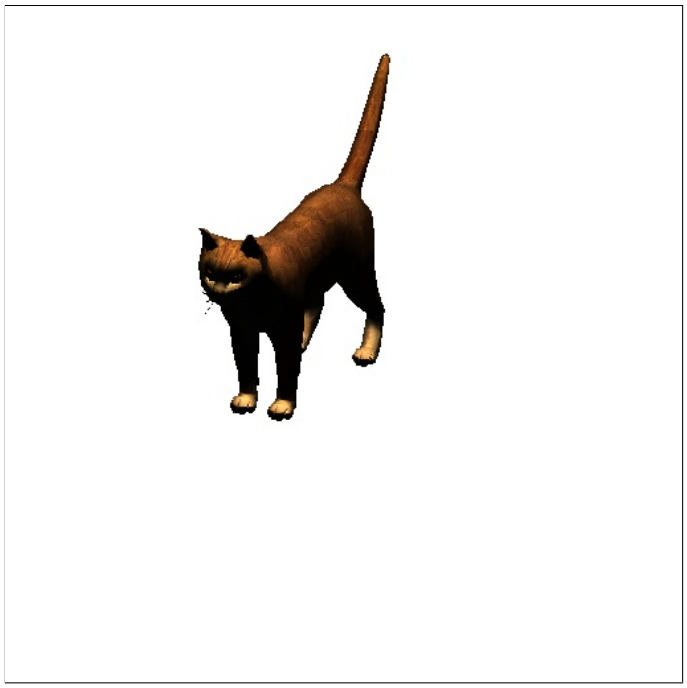

Figure 15: Phong Shading με μόνο την τρίτη πηγή και όλα τα είδη φωτισμού

10 Κάποιες έξτρα φωτογραφίες που παρουσιάζουν ενδιαφέρον

Ενδιαφέρον παρουσιάζει η χρήση της Gouraud Shading χωρίς να συμπεριληφθεί ο φωτισμός, με την απλή λογική των προηγούμενων παραδοτέων και το texture map. Το αποτέλεσμα φαίνεται στην εικόνα 16.

Επίσης στις εικόνες 17 και 18 μπορούμε να δούμε το αποτέλεσμα που υπήρχε πρίν την χρήση του texture map με Gouraud και Phong shading.

Figure 16: Gouraud Shading με texture map χωρίς να συμπεριληφθεί ο φωτισμός

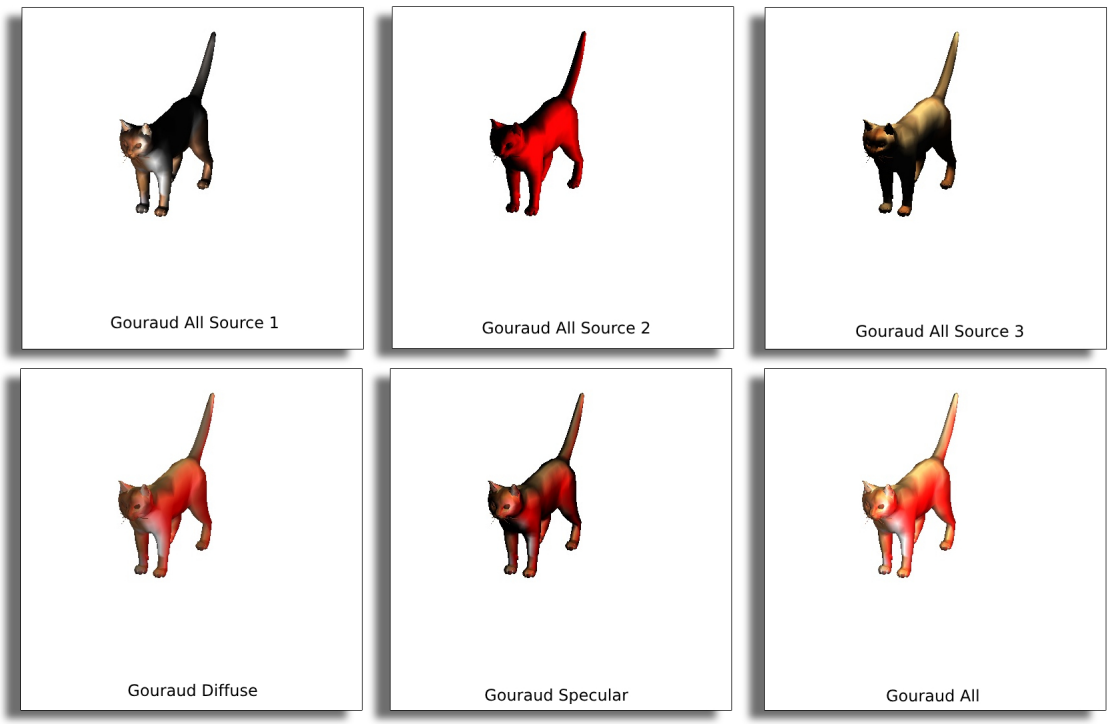

Figure 17: Gouraud Shading χωρίς texture map

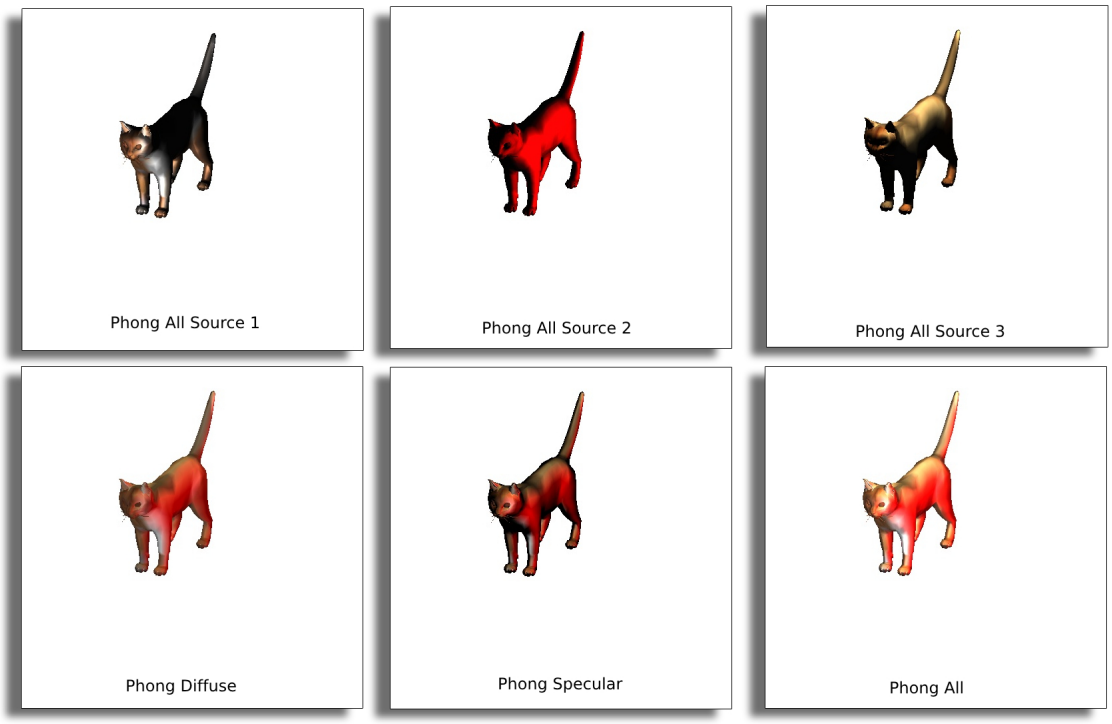

Figure 18: Phong Shading χωρίς texture map

11 Δομή Εργασίας

Κάθε μία από τις ζητούμενες συναρτήσεις καθώς και όσες χρησιμοποιούνται από προηγούμενα παραδοτέα βρίσκονται στο zip στο δικό τους αρχείο. Στο zip επίσης βρίσκεται το αρχείο `hw3.npy` με τα data και η `cat_diff.png` που χρησιμοποιείται ως texture map.

Στο zip δεν συμπεριλαμβάνονται οι φωτογραφίες, οι οποίες αποθηκεύονται κάθε φορά που τρέχει το `demo.py`, στον φάκελο στον οποίο βρίσκεται.

Τέλος έχουν συμπεριληφθεί δύο κολάζ φωτογραφιών που δείχνουν τα αποτελέσματα πριν την χρήση του texture map.