Sprawozdanie
Grafika komputerowa i komunikacja człowiek-komputer
Laboratorium 5

Katsiaryna Kolyshko 276708

Dr. Inż. Jan Nikodem

Wstęp teoretyczny

Zadanie laboratoryjne polega na teksturowaniu jajka. Musieliśmy dodać 3 tekstury.

Tekstury w 3D

Teksturowanie w modelowaniu 3D to krytyczny proces używany do dodawania szczegółowych powierzchni do modeli 3D. Polega on na stosowaniu obrazów (tekstur) lub wzorów proceduralnych do geometrii 3D, aby wyglądała bardziej realistycznie lub stylowo. Ten krok poprawia atrakcyjność wizualną modelu poprzez symulację wyglądu materiałów, takich jak drewno, metal, tkanina lub skóra. Tekstury mogą być malowane ręcznie lub tworzone przy użyciu fotografii i są mapowane na powierzchni modelu 3D.

Teksturowanie wykonuje się w celu osiągnięcia realizmu, zwiększenia estetyki i zapewnienia ważnych szczegółów powierzchni, które w przeciwnym razie wymagałyby intensywnego modelowania. Jest to szczególnie ważne w celu nadania głębi i szczegółów powierzchni, takich jak zadrapania, brud lub złożone wzory, bez dodawania większej ilości geometrii. Jest ono stosowane w różnych dziedzinach, takich jak gry, animacja, rzeczywistość wirtualna i projektowanie produktów, aby obiekty 3D wyglądały realistycznie lub przekazywały określony styl.

Zalety teksturowania

Realizm: Teksturowanie jest kluczem do nadania modelowi bardziej realistycznego wyglądu poprzez naśladowanie interakcji światła z powierzchniami. Może ono ukazywać niedoskonałości powierzchni, zwietrzenie i inne drobne szczegóły, których nie można przedstawić wyłącznie za pomocą geometrii.

Optymalizacja: Tekstury umożliwiają tworzenie skomplikowanych szczegółów bez konieczności zwiększania liczby wielokątów. Dzięki temu modele są lekkie i przyjazne dla wydajności, szczególnie w aplikacjach czasu rzeczywistego, takich jak gry wideo lub VR.

Kontrola kreatywna: Teksturowanie daje artystom możliwość dostosowywania materiałów, wzorów i jakości powierzchni. Za pomocą map, takich jak mapy wypukłości, mapy normalne i mapy lustrzane, można symulować efekty świetlne, dodawać głębię i kontrolować połysk lub chropowatość powierzchni.

Efektywność czasowa: Zamiast modelować każdy pojedynczy szczegół, teksturowanie pozwala artystom tworzyć złożone i bogate wizualnie modele w krótszym czasie, nakładając różne mapy tekstur na pojedynczy obiekt 3D.

Style artystyczne: Tekstury zapewniają większą swobodę w osiąganiu szerokiej gamy stylów artystycznych, od hiperrealistycznych tekstur po bardziej abstrakcyjne lub stylizowane projekty.

Wady teksturowania

Złożoność: Podczas gdy teksturowanie może być ogromną oszczędnością czasu, może być również skomplikowane i czasochłonne, szczególnie gdy próbujesz tworzyć bezszwowe tekstury lub upewnić się, że tekstury idealnie się pokrywają w całym modelu. Wymaga umiejętności w zakresie rozpakowywania UV, edycji obrazów i tworzenia materiałów.

Wymagające dużej ilości zasobów: Wysokiej jakości tekstury mogą zajmować znaczną ilość miejsca na dysku i pamięci. Duże tekstury mogą powodować problemy z czasem ładowania, szczególnie w grach i animacjach z wieloma zasobami.

Mapowanie bezszwowe: Jednym z wyzwań teksturowania jest zapewnienie, że tekstury wyglądają bezszwowo. Jeśli nie zostaną odpowiednio obsłużone, tekstury mogą wydawać się rozciągnięte lub źle wyrównane, co może zaburzyć immersję w animowanej scenie lub świecie gry.

Ograniczenia płaskich tekstur: Podczas gdy tekstury mogą symulować głębię i szczegółowość, nie mogą w rzeczywistości modyfikować geometrii modelu. W niektórych przypadkach może to prowadzić do nierealistycznych lub mniej wiarygodnych wyników, jeśli tekstura nie pasuje wystarczająco dobrze do podstawowej geometrii.

Teksturowanie vs kolorowanie

Podczas gdy kolorowanie polega po prostu na nałożeniu jednolitego koloru na obiekt, teksturowanie dodaje złożoności poprzez nałożenie szczegółowych, często wielowarstwowych powierzchni na model. Kolorowanie jest zazwyczaj stosowane w przypadku prostszych lub stylizowanych modeli, w których szczegółowość powierzchni jest minimalna, podczas gdy teksturowanie jest stosowane w przypadku bardziej złożonych i realistycznych przedstawień. Tekstury mogą obejmować mapy wypukłości, mapy normalne, mapy lustrzane i mapy rozproszone, które wszystkie działają razem, aby symulować rzeczywiste materiały i zachowanie światła. Kolorowanie z natury nie zapewnia tego samego poziomu szczegółowości powierzchni co teksturowanie.

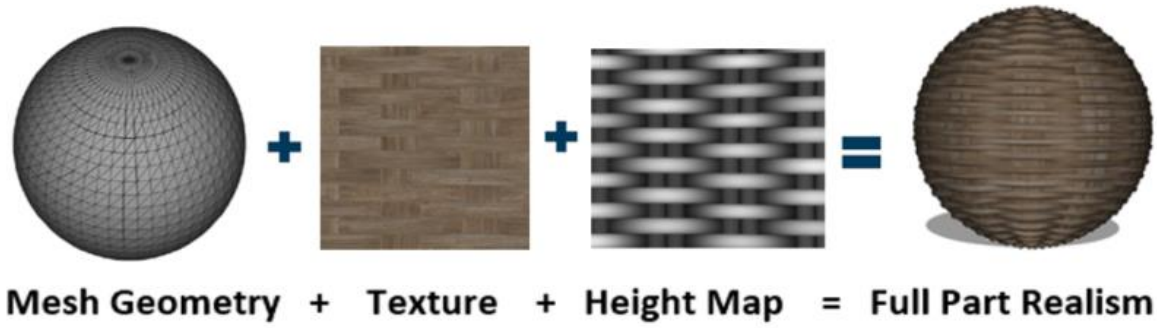
Teksturowanie w animacji i modelowaniu

W animacji tekstury są ważne nie tylko dla modelowania postaci i otoczenia, ale także dla przekazywania ruchu i interakcji ze światłem. Na przykład tekstura skóry animowanej postaci może się zmieniać w zależności od oświetlenia lub akcji (np. spocona postać w intensywnej scenie). W otoczeniu tekstury mogą być animowane, aby symulować takie rzeczy, jak warunki pogodowe, ruch wody lub efekty środowiskowe.

W przypadku modelowania 3D teksturowanie jest niezbędne, aby nadać modelowi „realny” wygląd lub stylizowany projekt. Modele tworzone na potrzeby animacji lub gier wideo opierają się na teksturach, aby określić, jak materiały, takie jak ubrania, skóra lub zbroja wyglądają i zachowują się w świetle. W modelowaniu celem jest stworzenie geometrii, która może prawidłowo oddziaływać z teksturami — wiąże się to z ostrożnym rozwijaniem UV, aby upewnić się, że tekstury są prawidłowo stosowane.

Ale aby otrzymać realistyczną teksturę, nie wystarczy po prostu nałożyć na model foto.

Aby uzyskać bardziej realistyczny wygląd w renderowaniu 3D, niezbędne jest połączenie geometrii siatki, mapowania tekstur i map wysokości. Geometria siatki definiuje podstawowy kształt i strukturę obiektu, zapewniając podstawę dla fizycznej formy modelu. Jednak sama geometria nie jest w stanie uchwycić drobnych szczegółów powierzchni, takich jak zmarszczki, zadrapania lub wzory, bez znacznego zwiększenia liczby wielokątów, co może być kosztowne obliczeniowo. Tekstury dodają szczegółów wizualnych poprzez stosowanie obrazów lub wzorów proceduralnych do powierzchni, symulując wygląd materiałów, takich jak drewno, metal lub tkanina. Aby jeszcze bardziej zwiększyć realizm, mapy wysokości (lub mapy przemieszczeń) modyfikują geometrię w czasie renderowania, przemieszczając wierzchołki na podstawie wartości skali szarości, tworząc głębię i skomplikowane cechy powierzchni, takie jak rowki lub wytłoczenia. Razem te elementy tworzą równowagę między wiernością wizualną a wydajnością, umożliwiając obiektom wydawanie się bardzo szczegółowymi i realistycznymi bez przytłaczania zasobów obliczeniowych.



Rysunek 1: Proces otrzymania realistyczniejszej tekstury

Opis działania programu

Nowe instrukcje biblioteki OpenGL:

1. *void glTexImage2D( )*

Definiuje dwuwymiarową teksturę. Argumenty pozwalają określić: typ tekstury, poziom szczegółowości, liczbę komponentów koloru, szerokość obrazu, wysokość, szerokość obramowania, format koloru, typ danych i wskaźnik do danych pikseli dla tekstury.

2. *void glTexCoord2f( )*

Określa współrzędne tekstury dla bieżącego wierzchołka. Argumenty s i t reprezentują współrzędne w przestrzeni tekstury, gdzie s jest współrzędną poziomą, a t jest współrzędną pionową.

3. *void glNormal3f( )*

Definiuje wektor normalny dla bieżącego wierzchołka. Argumenty nx, ny i nz reprezentują składowe wektora normalnego wzdłuż osi x, y i z, odpowiednio, pomagając OpenGL obliczyć efekty świetlne.

4. void glSwapBuffers(void)

Zamienia bufory przedni i tylny w renderowaniu z podwójnym buforowaniem, wyświetlając bieżącą klatkę na ekranie podczas przygotowywania następnej klatki w tle. Jest to używane w systemach okienkowych z podwójnym buforowaniem, aby zapobiec migotaniu.

Modyfikacje kodu.

Po pierwsze, pozbawiliśmy trybu modelowania jajka na punktach i liniach, bo, oczywiście, na nich nie będzie widoczna tekstura. Tak samo zmieniliśmy jak pada światło na sztywno i jest białe, 100% ambient.

Zaczniemy od definiowania zmiennych które będą potrzebne do programu:

```
46    texture_btn_z = 0  # Toggle between textures
47    image1 = Image.open("d7Txtr.tga")  # First texture image
48    image2 = Image.open("eggTxtr.tga")  # Second texture image
49    image3 = Image.open("cat.tga")
50
51    # Matrix for texture coordinates
52    matrixWithTextures = np.zeros((n + 1, n + 1, 2))
53    matrixWithVectors = np.zeros((n + 1, n + 1, 3))
```

Te zmienne służą do zarządzania mapowaniem tekstur i renderowaniem obiektu 3D w aplikacji graficznej. `texture_btn_z` to przełącznik umożliwiający przełączanie się między różnymi teksturami ('image1', 'image2', 'image3'), które są ładowane z plików TGA. 'matrixWithTextures' to macierz przechowująca współrzędne tekstury w celu mapowania wybranej tekstury na obiekcie 3D, zapewniając prawidłowe wyrównanie tekstury. Tymczasem 'matrixWithVectors' przechowuje dane pozycyjne 3D dla obiektu, łącząc geometrię z zastosowanymi teksturami podczas renderowania.

Funkcja matrixTextures():

```
56    def matrixTextures():  3 usages  new *
57        """Calculate texture coordinates for the egg surface"""
58        global matrixWithTextures
59        # Resize the matrix based on the current value of n
60        matrixWithTextures = np.zeros((n + 1, n + 1, 2))
61
62        for i in range(0, n + 1):
63            for j in range(0, n + 1):
64                u = i / n
65                v = j / n
66
67                # Rotate texture on the proper half
68                if i > (n / 2):
69                    matrixWithTextures[i][j][0] = v
70                    matrixWithTextures[i][j][1] = 1 - 2 * u
71                else:
72                    matrixWithTextures[i][j][0] = v
73                    matrixWithTextures[i][j][1] = 2 * u
```

Funkcja 'matrixTextures()' oblicza współrzędne tekstury w celu odwzorowania tekstury na powierzchni jajowatego obiektu 3D. Najpierw inicjuje 'matrixWithTextures' jako macierz 3D o rozmiarze opartym na rozdzielczości 'n'. Następnie dla każdego punktu (i, j) na powierzchni jajka oblicza współrzędne tekstury (u, v) znormalizowane do zakresu od 0 do 1. W zależności od tego, czy punkt znajduje się w górnej czy dolnej połowie jajka, dostosowuje orientację tekstury, odpowiednio modyfikując współrzędne 'u' i 'v'. Zapewnia to właściwe wyrównanie i symetrię tekstury na powierzchni jajka.

Funkcja matrixWithVectorsValues( ):

Funkcja 'matrixWithVectorsValues()' oblicza wektory normalne dla każdego punktu na powierzchni jajowatego obiektu 3D. Inicjuje 'matrixWithVectors' jako macierz 3D o rozmiarze zgodnym z n. Dla każdego punktu (i, j) oblicza pochodne cząstkowe (xu, xv, yu, yv, zu, zv) na podstawie równań parametrycznych powierzchni jajka. Te pochodne są używane do obliczenia iloczynu wektorowego, co daje składowe (x, y, z) wektora normalnego. Wektor jest normalizowany do długości jednostkowej i odwracany dla punktów w dolnej połowie jajka, aby zapewnić spójną orientację. Na koniec znormalizowane składowe są przechowywane w 'matrixWithVectors' do obliczeń renderowania i cieniowania.

Parametry (xu, xv, yu, yv, zu, zv) reprezentują pochodne cząstkowe powierzchni parametrycznej jajka względem parametrów u i v. Są one potrzebne, ponieważ pozwalają nam obliczyć wektory styczne do powierzchni, które są kluczowe dla określenia wektorów normalnych.

xu, yu, zu: Częściowe pochodne powierzchni jajka w kierunku u. Opisują one, jak powierzchnia zmienia się, gdy u się zmienia, utrzymując stałe v. Zasadniczo dają wektor styczny w kierunku u.

xv, yv, zv: Częściowe pochodne powierzchni jajka w kierunku v. Opisują one, jak powierzchnia zmienia się, gdy v się zmienia, utrzymując stałe u. Dają wektor styczny w kierunku v.

Obliczając iloczyn wektorowy tych dwóch wektorów stycznych, wyprowadzamy wektor normalny, który jest prostopadły do powierzchni w każdym punkcie.

```python
def matrixWithVectorsValues():  3 usages  new *
    """Calculate normal vectors for the egg surface"""
    global matrixWithVectors
    # Resize the matrix based on the current value of n
    matrixWithVectors = np.zeros((n + 1, n + 1, 3))

    for i in range(0, n + 1):
        for j in range(0, n + 1):
            u = i / n
            v = j / n

            xu = (-450 * pow( *args: u, 4) + 900 * pow( *args: u, 3) - 810 * pow( *args: u, 2) + 360 * u - 45) * cos(pi * v)
            xv = pi * (90 * pow( *args: u, 5) - 225 * pow( *args: u, 4) + 270 * pow( *args: u, 3) - 180 * pow( *args: u, 2) + 45 * u) * sin(pi * v)
            yu = 640 * pow( *args: u, 3) - 960 * pow( *args: u, 2) + 320 * u
            yv = 0
            zu = (-450 * pow( *args: u, 4) + 900 * pow( *args: u, 3) - 810 * pow( *args: u, 2) + 360 * u - 45) * sin(pi * v)
            zv = (- pi) * (90 * pow( *args: u, 5) - 225 * pow( *args: u, 4) + 270 * pow( *args: u, 3) - 180 * pow( *args: u, 2) + 45 * u) * cos(pi * v)

            x = yu * zv - zu * yv
            y = zu * xv - xu * zv
            z = xu * yv - yu * xv

            sum = pow( *args: x, 2) + pow( *args: y, 2) + pow( *args: z, 2)
            length = sqrt(sum)

            if length > 0:
                x = x / length
                y = y / length
                z = z / length

            if i > n / 2:
                x *= -1
                y *= -1
                z *= -1

            matrixWithVectors[i][j][0] = x
            matrixWithVectors[i][j][1] = y
            matrixWithVectors[i][j][2] = z
```

Funkcja set_texture():

Funkcja 'set_texture()' przypisuje i ustawia aktywną teksturę do renderowania na podstawie bieżącej wartości 'texture_btn_z', która przełącza się między trzema teksturami. W zależności od stanu przełączania wybiera odpowiedni obraz i przesyła jego dane pikselowe do GPU jako teksturę 2D przy użyciu funkcji OpenGL 'glTexImage2D()'. Dane tekstury są przekazywane w surowym formacie RGB, a wymiary obrazu definiują rozdzielczość tekstury. Ta funkcja umożliwia dynamiczne przełączanie między teksturami podczas renderowania, umożliwiając interaktywną zmianę wyglądu obiektu 3D.

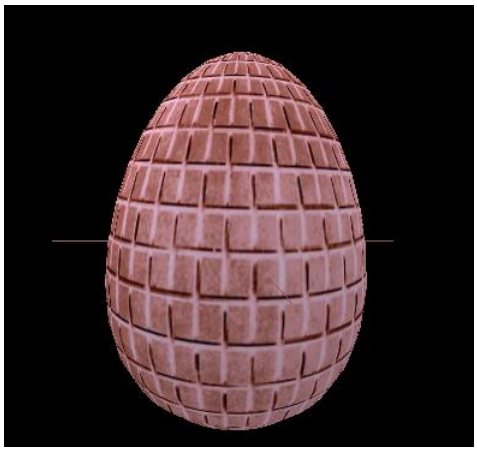
```python
def set_texture():  1 usage  new *
    """Set the current texture based on the toggle state"""
    if texture_btn_z == 0:
        img = image1
    elif texture_btn_z == 1:
        img = image2
    else:  # texture_btn_z == 2
        img = image3

    glTexImage2D(
        GL_TEXTURE_2D,  level: 0,  internalformat: 3, img.size[0], img.size[1],  border: 0,
        GL_RGB, GL_UNSIGNED_BYTE, img.tobytes( encoder_name: "raw",  *args: "RGB", 0, -1)
    )
```

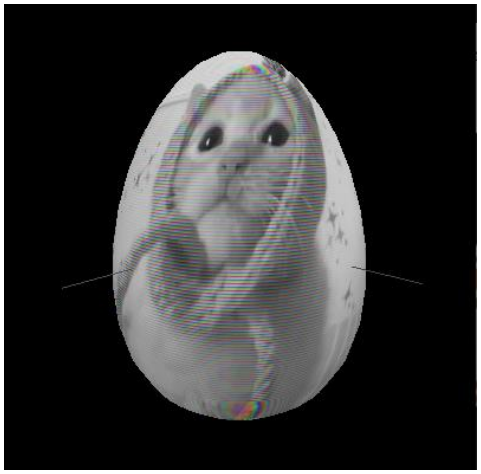
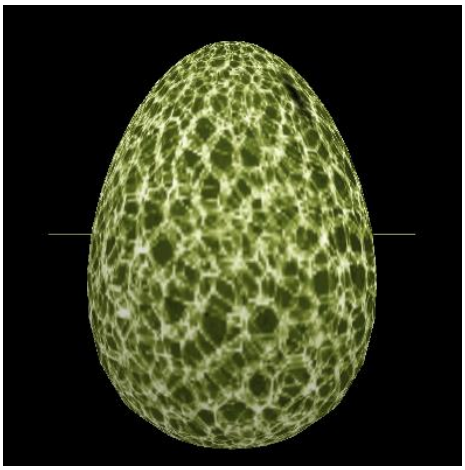
Funkcja keyboard_key_callback( ):

```
326     if key == GLFW_KEY_Z and action == GLFW_PRESS:
327         # Cycle through 0, 1, and 2
328         texture_btn_z = (texture_btn_z + 1) % 3  # 0 -> 1 -> 2 -> 01
```

Dodaliśmy nowy przycisk, że aby zmienić teksturę – musimy nacisnąć klawisze ‘z’.

Wyniki działania programu:



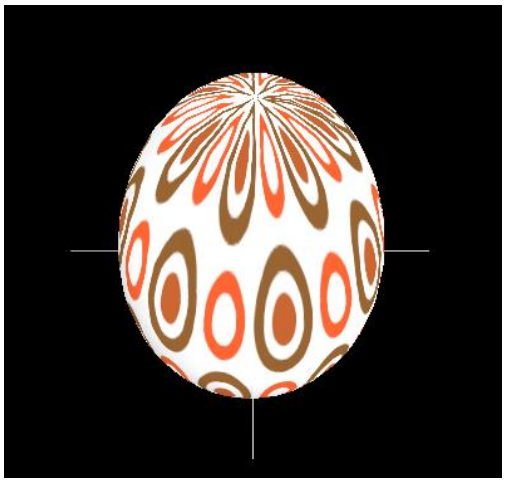



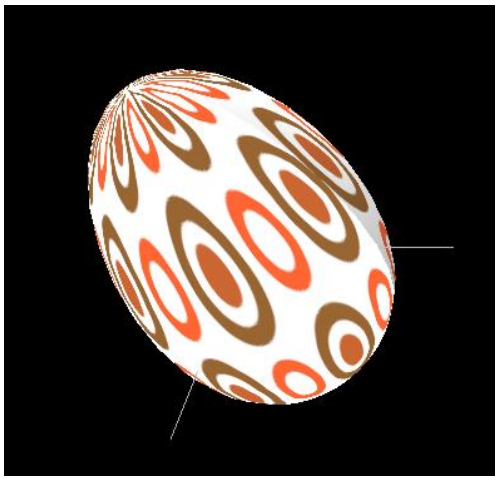
Wnioski:

Kiedy pracowałam nad kodem, miałam problem z tym, że teksturowanie nie przechodzi do końca poprawnie. Naprzykład, widać to będzie na następnym foto:

Spróbowałam rozwiązać ten problem zmieniając funkcje matrixTextures() i matrixWithVectorsValues()

Ale otrzymałam następny wynik: pozbawiłąm się od tego problemu na jajku, ale teraz niepoprawnie wygląda tekstura:





Myślę, że musiałabym w cale przerobić jak nakłada się tekstura, aby poprawnie i ładnie nakładała się. Próbowałam jeszcze kilka sposobów, ale nie rozwiązało to mój problem, na żal.

Porównanie tekstur:

Obraz 1 to tekstura o niskiej szczegółowości, z prostym wzorem przypominającym cegłę. Tekstura ma stosunkowo niską zmienność pikseli i ograniczoną liczbę unikalnych kolorów, co sugeruje, że byłaby efektywna do renderowania w aplikacjach w czasie rzeczywistym.

Obraz 2 ma teksturę o średniej szczegółowości, z powtarzającym się wzorem okrągłych kształtów w różnych odcieniach brązu i pomarańczy. Ta tekstura wykazuje wyższy poziom szczegółowości i złożoności kolorów w porównaniu ze wzorem cegły, co może skutkować zwiększonymi wymaganiami dotyczącymi renderowania.

Obraz 3 ma teksturę o wysokiej szczegółowości, z czymś, co wygląda jak zniekształcony i pełen zakłóceń obraz kota. Ta tekstura ma dużą liczbę unikalnych kolorów i znaczną zmienność pikseli, co wskazuje na wysoki poziom złożoności. Renderowanie tego typu szczegółowej, fotorealistycznej tekstury prawdopodobnie wymagałoby większych zasobów obliczeniowych i zaawansowanych technik mapowania, aby zachować wierność wizualną.

Podsumowując, poziom szczegółowości tekstury ma bezpośredni wpływ na złożoność i wymagania dotyczące wydajności renderowania. Tekstury o niskiej szczegółowości, takie jak wzór cegieł, można wydajnie przetwarzać i mapować, podczas gdy tekstury o wysokiej szczegółowości, takie jak obraz kota, wymagają bardziej zaawansowanych technik renderowania i zasobów obliczeniowych, aby utrzymać jakość wizualną. Zrozumienie tych kompromisów jest kluczowe przy wyborze i optymalizacji tekstur do aplikacji w czasie rzeczywistym.

Inna interesująca rzecz to jak wygląda tekstura kotka. Oryginalnie, tekstura jest kolorowana, ale na jajku jest czarno-biała. Ale kiedy obracałam to jajko, to było widać jeszcze jakieś kolorki. Co to za efekt? Odpowiedź: aberracja chromatyczna.

Aberracja chromatyczna to zjawisko optyczne występujące, gdy soczewka nie jest w stanie skupić wszystkich kolorów światła widzialnego w tym samym punkcie zbieżności. Powoduje to przemieszczenie różnych długości fal kolorów, tworząc tęczową obwódkę wokół krawędzi o wysokim kontraście na obrazie.

W przypadku tekstury kota na jajku, zakrzywiona powierzchnia jajka działa jak soczewka. Gdy światło odbija się od nierównej tekstury, krzywizna jajka powoduje, że różne długości fal kolorów lekko się rozdzielają, tworząc efekt tęczy, który widzisz.

Jest to powszechny problem w przypadku obiektywów niskiej jakości lub czujników obrazu, ponieważ nie są one w stanie idealnie skupić wszystkich długości fal kolorów. W grafice komputerowej aberrację chromatyczną można symulować, aby dodać realizmu i niedoskonałości do renderowanych obrazów, ale ogólnie jest ona uważana za niepożądany artefakt w obrazowaniu wysokiej jakości.

Powód, dla którego oryginalny obraz kota wydaje się kolorowy, podczas gdy tekstura na jajku wygląda na czarno-białą, jest prawdopodobnie spowodowany procesem wypiekania tekstury na trójwymiarowym modelu jajka. W trakcie tego procesu szczegóły kolorów o wysokiej częstotliwości mogły zostać utracone lub spłaszczone, co spowodowało monochromatyczny wygląd powierzchni jaja.