Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Markus Gross
Remo Ziegler, Christian Voegeli, Daniel Cotting

# Visual Computing
# Mipmaps

## Ziele

Vertiefung der Vorlesung in den Bereichen Texturierung, Mipmaps und Filterung.

## Grundlagen

In der perspektivischen Ansicht verändert sich die Grösse eines Objektes mit dem Abstand des Objekts zum Betrachterstandpunkt. Bei der Texturierung von Objekten führt dies im Allgemeinen zu einer Unterabtastung der Textur und somit zu Aliasing-Effekten (siehe Skript Kapitel 11). Um dieses Problem zu lösen, werden je nach Darstellungsgrösse des Objektes Texturen unterschiedlicher Auflösungsstufen verwendet, die mittels einer Filterung aus der Originaltextur abgeleitet werden.

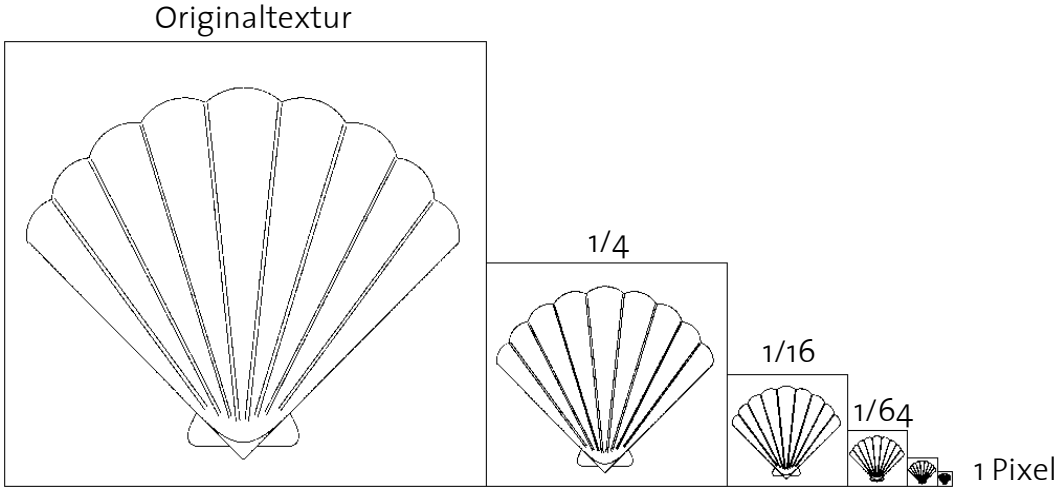


Bild 1: Verschiedene Auflösungsstufen einer Textur (Mipmaps).

## 1) Mipmap-Texturierung

Zur Visualisierung des Mipmap-Verfahrens soll eine Mipmap konstruiert werden, welche jeden Mipmap-Level mit einer anderen Farbe darstellt.
In der vorgegebenen Funktion `generateMipMaps()` zur Erstellung einer Mipmap wird für jeden Mipmap-Level die Funktion `coloredMipMap(level)` aufgerufen um die entsprechende Mipmap-Textur zu definieren, wobei der Aufruf zur Definition der Basis-Textur (`level=0`) bereits in `initTexturing()` erfolgt.

a) Implementieren Sie die Funktion `coloredMipMap(level)`, welche für einen gegebenen Mipmap-Level die Farbe der Textur bestimmt und die Textur in OpenGL definiert. Benutzten Sie dazu die OpenGL-Funktion `glTexImage2D(...)`, deren Anwendung an verschiedenen Stellen im Code bereits illustriert ist.

**Anmerkungen**

- Die Dimension der quatratischen Basistextur ist durch TEXTURE_SIZE definiert.
- Um für verschiedene Texturgrössen den selben Datentyp verwenden zu können, bedienen wir uns eines linearen Arrays, dessen Grösse für die maximale Anzahl Texturpixel ausgelegt ist, und benutzen für kleinere Texturen nur Teilbereich des Arrays.
- Die Struktur `Pixel` enthält die Rot-, Grün- und Blau-Komponente eines Pixels als `GLubyte`-Werte.

b) Aktivieren Sie nun die ***Darstellung der Mipmap-Texturen***, indem Sie am Ende der Funktion `initTexturing()` die zu verwendende Mipmap-Filterung definieren und das Texturieren der Objekte einschalten.

c) Führen Sie Ihr Programm aus und beobachten Sie die ***Wirkung der Mipmap-Texturierung*** in Kombination mit den verschiedenen Texturen und verschiedenen Mipmap-Filtern. (Übersicht der Tastaturbelegung wird auf die Konsole ausgegeben)

## 2) Filterung bei der Mipmap-Generierung

Die in Aufgabe 1 angewandten Filterfunktionen betreffen nur die Abbildung der Texturen vom Textur- in den Objektraum. Aliasing-Effekte treten aber bereits bei der Generierung der Mipmap-Texturen aus der Originaltextur auf. Um die visuelle Qualität der Texturierung weiter zu steigern, sollen bei der Mipmap-Generierung anstelle des trivialen "nearest neighbor"- Ansatzes bessere Filter zum Einsatz kommen.

a) Implementieren Sie die Funktion `boxFilter(level)`, welche für eine Textur mittels einem Box-Filter die Mipmap zu dem gegebenen Level berechnet. Orientieren Sie sich dabei an der im Sourcecode bereits vorgegebenen "nearest neighbor"-Filterfunktion (`nearestNeighborFilter`).

**Anmerkungen**

- Die Basis Textur (`level = 0`) wurde bereits an einer anderen Stelle im Code definiert und in einer Datenstruktur zur Speicherung der Mipmap-Pyramide abgelegt.
- Benutzen Sie die vorgegebene Funktion `Pixel* getMipMapPixel(x, y, level)`, um Pixelwerte aus bereits definierten Mipmap-Levels auszulesen.

b) Implementieren Sie die Funktion `bilinearFilter(level)`, welche anstelle des Box Filters die angegebene bilineare Filtermatrix verwendet. Bauen Sie Ihre Lösung auf den Code des Box Filters auf.

$$\frac{1}{81}\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

## 3) Anisotropische Filterung & OpenGL Extensions

Trilineares Mip-Mapping kann in Extremfällen zu unscharfer Texturierung führen. Dies tritt dann auf, wenn die quadratische Textur auf ein stark degeneriertes Rechteck gemappt wird. Dem kann mit anisotroper Filterung entgegengewirkt werden. Diese berücksichtigt diesen speziellen Fall indem sie die Textur entlang der gestreckten Richtung mehrfach abgreift.
Um einen anisotropen Filter in unser Programm einzubauen, verwenden wir die OpenGL Extension `GL_EXT_texture_filter_anisotropic`.
OpenGL Extensions sind Erweiterungen zum OpenGL-Standard in Form von Spezifikationen. Diese werden vom OpenGL-Konsortium anerkannt und registriert. Die Implementierung davon

bleibt den Anbietern überlassen. Es ist deshalb jeweils abhängig vom System, welche Extensions unterstützt werden, und welche nicht.

a) Stellen Sie Programmbeginn fest, ob `GL_EXT_texture_filter_anisotropic` von ihrem System unterstützt wird. `glGetString(GL_EXTENSIONS)` gibt Ihnen eine Liste aller unterstützten Extensions in Form eines Strings zurück. Speichern Sie das Ergebniss in der globalen (boolean) Variable `anisotropic_supported`.

b) Aktivieren Sie den anisotropen Filter in der Funktion initTexturing() mit der Codezeile `glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, maxAnisotropy)`.