Computergrafik 2

Lab 2

Vorbereitungen

1. Kopieren Sie die Datei TextureMap.js, TriangleMeshGL.js, Mesh3D.js, Matrix4.js und Vec3.js, sowie die Shader vom letzten Lab in die Verzeichnisstruktur des neuen Labs
2. Kopieren Sie den Ordner RustMixedOnPaint in den Ordner data
3. Kopieren Sie den Methodenrumpf aus der Datei computeTangentFrame_stub.js in die Datei TriangleMeshGL.js.

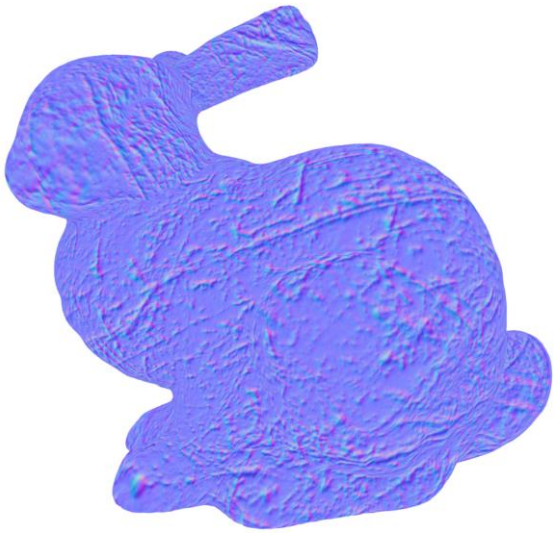
Aufgabe 1 Normal-Maps Samplen und im Shader entpacken

Laden Sie im Mesh3D.js die Texturen `"./../../data/RustMixedOnPaint/Diffuse.png"` und `"./../../data/RustMixedOnPaint/NormalMap.png`.

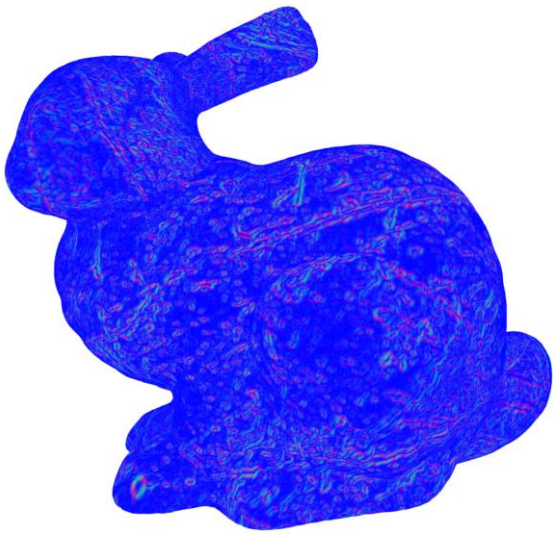
a) Modifizieren Sie zum Testen das Programm so, dass die diffuse Farbe von der diffusen Textur gelesen wird. Sie sollten dann dieses Ergebnis erhalten:

Hinweis: Das Feature vom letzten Lab, welches zwei Texturen mischt, brauchen wir in dieser Aufgabe nicht und Sie können es entfernen.

b) Samplen Sie nun den Inhalt er Normal-Map-Textur und geben Sie diesen aus. Sie sollten dann dieses Ergebnis erhalten:



c) Die Komponenten der Normal-Map liegen jedoch zwischen $[0\ldots1]$. Transformieren Sie die Komponenten nun so, dass diese zwischen $[-1\ldots1]$ liegen! Normalisieren Sie den resultierenden Normalenvektor. Geben Sie den Absolut-Betrag aus und erfreuen Sie sich dann an folgendem Ergebnis:



Aufgabe 2 Tangent-Frame berechnen

Für Tangent-Space Normal-Mapping benötigt man einen Tangent-Space. Diesen sollen Sie in der Methode `computeTangentFrame(triangles, positions, textureCoordinates)` der Klasse `TriangleMeshGL` berechnen. Diese Methode soll Tangenten (`outTangents`) und Bitangenten (`outBiTangents`) für jeden Vertex berechnen.

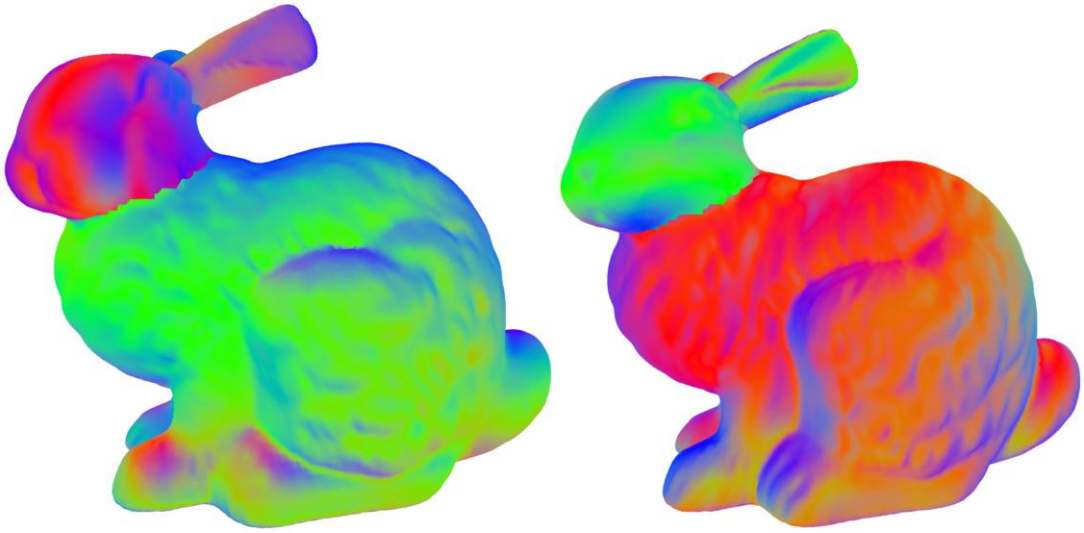
a) Im Code-Stub werden zum Testen `outTangents` und `outBiTangents` zunächst auf feste (aber falsche) Werte gesetzt. Erstellen Sie für diese beiden Arrays einen WebGL Array-Buffer und hängen Sie diese ans Vertex-Array-Objekt. Leiten Sie die Attribute vom Vertex-Shader an den Fragment-Shader weiter und geben Sie diese zum Testen als Farbe aus. Für die `outTangents/outBiTangents` sollten Sie einen komplett roten bzw. grünen Hasen sehen.

b) Berechnen Sie nun in der Methode `computeTangentFrame` für jeden Vertex die normalisierte Tangente und Bitangente. Mitteln Sie dazu die Tangenten und Bitangenten der Dreiecke, die an einem Vertex hängen. Dabei müssen Sie für jedes Dreieck die Tangente und Bitangente bestimmen (vgl. Folie 43) und auf die am Dreieck beteiligten Tangenten/Bitangenten der Vertices addieren. Abschließend müssen Sie die per-Vertex Tangenten/Bitangenten noch normalisieren.

Hinweis: Das Verfahren läuft sehr ähnlich wie bei der Berechnung der per-Vertex Normalen vom letzten Semester.

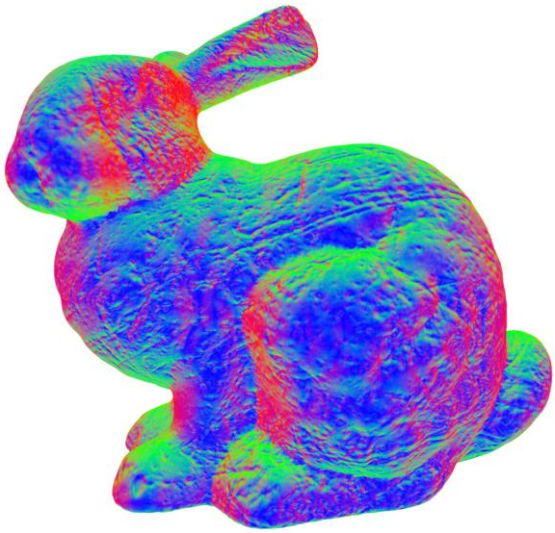
Hinweise: Bei der Berechnung der Tangenten und Bitangenten je Dreieck benötigen Sie die inverse einer 2x2 Matrix. Ein paar Helfermethoden finden Sie in der Datei Vec2.js.

Belohnen Sie sich mit folgenden Bildern, in dem Sie im Fragment-Shader die Absolutbeträge der Tangenten (linkes Bild) und Bitangente (rechtes Bild) ausgeben:



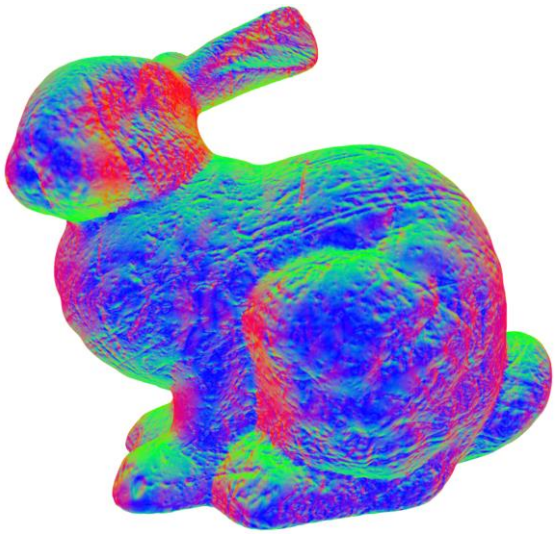
Aufgabe 3 Tangent-Frame Normal Maps

a) Transformieren Sie die von der Normal-Map-Textur gesampelte Normale vom Tangent-Space in den Objekt-Raum. Wenn Sie zum Debuggen den Absolut-Betrag der Normale ausgeben, könne Sie stolz dieses Bild betrachten:

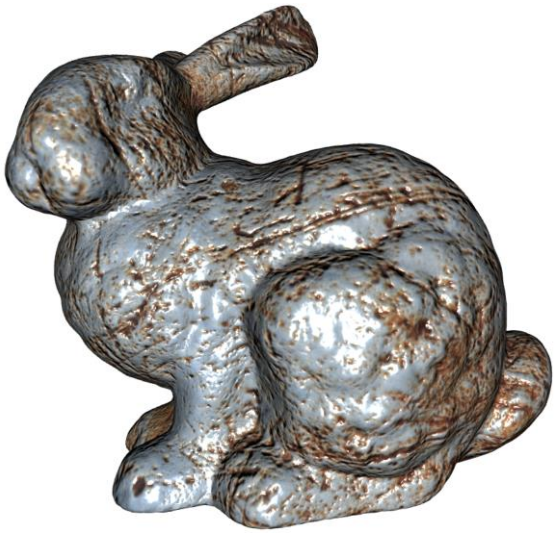


Hinweise: Sie benötigen die GLSL Funktion `transpose` und `inverse`.

b) Orthonormaliseren Sie vor dem Samplen noch den Tangent-Frame. Sie sollten folgendes pittoreskes Ergebnis Bild erhalten:



c) Nehmen Sie nun die in den Objekt-Raum transformierte Normale für die Beleuchtungsberechnung her. Samplen Sie zusätzlich die diffuse Farbe aus der Textur um nun das finale Ergebnis zu bekommen:



d) *Bonus:* Sie können zum Testen die beiden Check-Boxes Use Normal Mapping und Use Gram-Schmidt vom Fragment-Shader entsprechend verarbeiten lassen.