Inhaltsverzeichnis

1 Einleitung

Dieser Text beschreibt, wie man ein SIRDS oder SIS Hardware-beschleunigt rendern kann.

SIRDS steht dabei für *Single Image Random Dot Stereogram* und bezeichnet ein in einem einzelnen Bild enthaltenes Stereogramm, dass ohne weitere Hilfsmittel durch Schielen als 3D-Bild wahrgenommen werden kann. Random Dot bezieht sich darauf, dass die korrespondierenden Punktpaare, die vom Auge zu einem einzelnen 3D-Punkt fusioniert werden, zufällig aus der Menge aller möglichen korrespondierenden Punktpaare ausgewählt werden.

Historisch sind die *Single Image Stereograms* (SIS) eine Weiterentwicklung der SIRDS[1], bei denen nicht mehr nur Zufallspunkte gesetzt werden, sondern die Farben der korrespondierenden Punktpaare aus einem Basisbild gewählt werden, wodurch sich schönere und auch zur 3D-Szene passende Bilder erzeugen lassen.

Da SIRDS/SIS autostereoskopische Stereogramme sind, haben sie gegenüber den meisten anderen Renderverfahren den Vorteil, dass man sie ohne spezielle Hardware direkt als echtes 3D sehen kann.

Da es sich bei jedem SIRDS/SIS-Stereogramm um ein einzelnes Bild handelt, kann man sie in beliebiger Größe erstellen und die ganze Bildschirmauflösung ausnutzen. Außerdem wird es dadurch möglich, dass man große Szenen mit dem Parallelblick betrachten kann, bei dem die fokussierenden Augenmuskeln entspannt sind. [1]

Ein Nachteil ist, dass ein solches Stereogramm nur Entfernungsinformationen enthält, so dass keine Farben, Texturen, Lichter, Schatten, etc. dargestellt werden können, was bei anderen Stereogrammen (zwei Bilder nebeneinander) möglich ist. Man kann aber in ein fertiges SIRDS/SIS als *Floater* bezeichnete 2D-Sprites einblenden, die beliebige Texturen haben können.

Mein Programm erzeugt nur SIS-Bilder, ich schreibe aber im folgenden trotzdem SIRDS, da SIRDS den Spezialfall eines SIS mit einer Zufallstextur darstellt

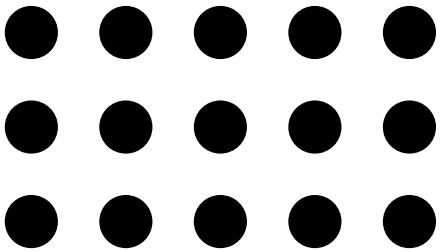
[1] Im Gegensatz zu Autostereogrammenarten, die aus zwei benachbarten Bildern bestehen, und ab einer gewissen Größe nur mit dem Kreuzblick gesehen werden können, bei dem man die fokussierenden Muskeln extrem anspannen muss. Die Anspannung in den akkumulierenden Muskeln ist zwar jeweils umgekehrt, aber da die meisten Leute heutzutage kurzsichtig sind, können viele ein SIRDS auch mit vollständig entspannten Muskeln betrachten.

und außerdem schöner klingt. (da man bei SIS immer an Silicon Integrated Systems Corp. denken muss...)

2 Grundlegender SIRDS/SIS-Algorithmus

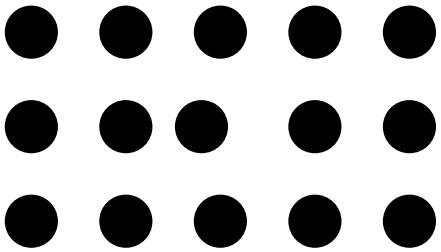
Dieser Abschnitt erklärt, wie die SIRDS funktionieren und orientiert sich an der Beschreibung von [2], von wo auch der verwendete Algorithmus stammt.

Dazu ist es zweckmäßig, sich ein Bild als Gitter von Punkten vorzustellen:

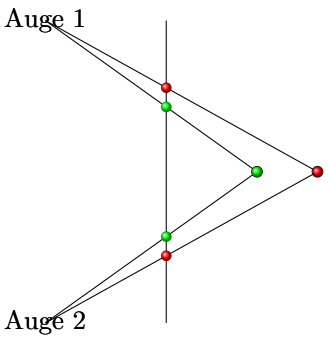


Betrachtet man das obige Gitter mit dem Parallelblick (mit Fokus hinter die Ebene des Texts) oder dem Kreuzblick (Fokus vor die Textebene), dann scheinen je zwei benachbarte Punkte zu einem einzelnen zu verschmelzen.[2]

Ein 3D-Effekt lässt sich nun einfach dadurch erreichen, dass man den Abstand zweier Punkte verändert:



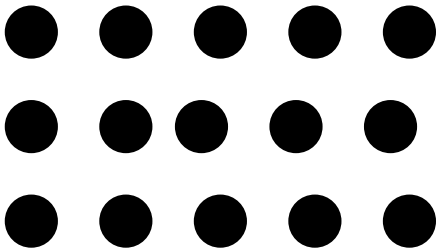
Dies liegt daran, dass das die beiden benachbarten 2D-Punkte als ein einzelner 3D-Punkt wahrgenommen werden, und das Gehirn daran gewohnt ist, dass unterschiedliche Positionen desselben Punktes auf beiden Netzhäuten durch den Abstand entlang der Blickrichtung verursacht werden:



[2]Die Punkte ganz links und ganz rechts werden von jeweils einem Auge als 2D-Punkte ohne Korrespondenzpunkt gesehen.

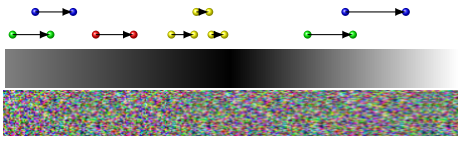
Am obigen Gitter mit dem verschobenen Gitterpunkt erkennt man, dass die Verringerung des Abstands beim Parallelblick zu der Illusion eines nahen Punkts und eine Vergrößerung zu der eines weit entfernten Punktes führt.[3]

Man sieht auch, dass nur der Abstand zum benachbarten Punkt, und nicht etwa die Position im Gitter eine Rolle spielt. Verschiebt man daher einen Punkt nach vorne (also nach links), müssen alle folgenden Punkte ebenfalls verschoben werden, um ihren z-Wert konstant zu halten. Das oben mit dem verschobenen Punkt gezeichnete Gitter, müsste also eigentlich folgendermaßen aussehen:



Da der Zusammenhang zwischen dem horizontalen Abstand im Bild und der im 3D-Raum wahrgenommener Distanz $z(x)$ ungefähr linear ist, kann man ihn als $d + z(x) \cdot f$ schreiben, wobei d der Basisabstand der Punkte ist, der bestimmt, wie sehr man schielen muss, um das SIRDS dreidimensional sehen zu können, weshalb d auch ungefähr dem Augenabstand entspricht. f ist ein Skalierungsfaktor, der das Verhältnis zwischen x-Achse und z-Achse festlegt[4][5].

Auf diese Weise lässt sich also ein SIRDS erzeugen, indem man alle Pixel eines Bildes entsprechend verschiebt. Da dies jedoch relativ kompliziert ist, geht man bei der tatsächlichen Erstellung eines SIRDS genau andersherum vor[2]: Man zeichnet das SIRDS von links nach rechts und ermittelt an jedem Pixel, dessen z-Wert und daraus den Abstand $(d + z \cdot f)$ zum vorherigen Pixel. Dann verschiebt man allerdings nichts, sondern kopiert einfach den Farbwert des $(d + z \cdot f)$ entfernten Pixels zum aktuellen Pixel:



Indem man das Bild so nach rechts durchläuft, erhält man in $O(\text{pixels})$ ein vollständiges SIRDS.

Ist das SIRDS fertig, kann man noch Floater einblenden, indem man einfach an einer Stelle ein 2D-Bild drüber zeichnet. Da aber schon dieses eine 2D-Bild doppelt gesehen wird, muss man das Bild noch im Abstand $(d + zf)$ links und rechts davon mehrmals wiederholen, so dass man eine Reihe von insgesamt $\frac{width}{d+zf}$ 2D-Bildern erhält.

[3] Beim Kreuzblick ist es entsprechend umgekehrt

[4] Für den Parallelblick ist $f > 0$, setzt man $f < 0$ wird das Bild ≫invertiert≪ und man kann den Kreuzblick benutzen

[5] In meinem Programm ist f bis auf das Vorzeichen konstant und man kann stattdessen das Bild insgesamt skalieren

3 Parallelisierung

Das Problem der Implementierung als Shader ist, dass der Shader alle Pixel parallel verarbeitet. Denn obgleich es einen großen Geschwindigkeitsgewinn bringt, alle Zeilen des SIRDS parallel erzeugen zu können, müssen die Pixel innerhalb derselben Zeile sequentiell gesetzt werden, da die Farbe eines Pixels von der Farbe der vorherigen Pixel abhängt.

Glücklicherweise ist es immer möglich, ganze Pixel-Streifen parallel zu zeichnen: Da der Abstand zu einem vorherigen Pixel $d + z \cdot f$ mit $z \in [z_{min}, z_{max}]$ ist, gibt es einen minimalen Abstand $d_{min} = d + z_{min} \cdot f$ und einen maximalen Abstand $d_{max} = d + z_{max} \cdot f$. [6]

Daraus folgt, dass ein Pixel keinen Pixel, in einem Abstand von weniger als d_{min} beeinflussen kann, und man somit in einer Zeile jeweils d_{min} benachbarte Pixel parallel rendern kann.

Allerdings ist es vor dem Zeichnen eines Pixels nötig, dass jeder Pixel, der diesen Pixel beeinflussen kann, bereits auf die richtige Farbe gesetzt worden ist. Also muss vor jedem neu gerenderten Pixel zumindest bereits ein Streifen der Breite $d_{max} - d_{min} = |d + z_{max} f - d - z_{min} f| = (z_{max} - z_{min})|f|$ erstellt worden sein. Da dieser Streifen aber gerade ausreicht, um einen Pixel zu setzen, ist es sinnvoller immer Blöcke der Länge d_{min} zu zeichnen, ohne vor den Blöcken Lücken zu lassen:

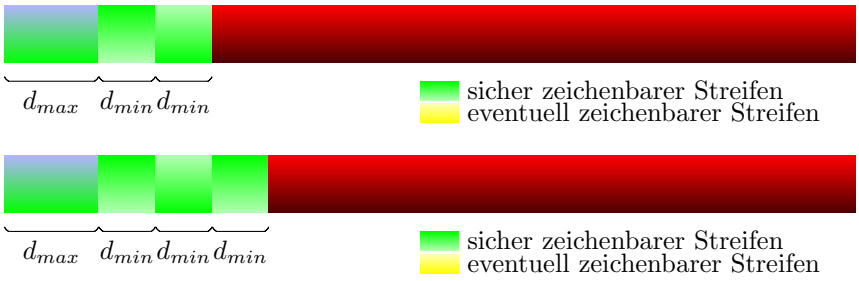
Im ersten Schritt zeichnet man also den Basisstreifen am linken Bildschirmrand, der die für das SIS gewünsche Textur enthält (dieser Streifen muss jedoch eine Länge von d_{max} haben, damit der Pixel direkt rechts davon gezeichnet werden kann[7]):

Anschließend lassen sich auf jeden Fall die Pixel in einem Streifen der Länge d_{min} parallel zeichnen, da der Referenzpixel jedes Pixel innerhalb diesem Streifen zum Basisstreifen gehört. In bestimmten Situationen, wenn der Depthbuffer maximale Werte enthält, lässt sich sogar ein Streifen der Länge d_{max} zeichnen:

Um zu wissen, ob der neue Streifen eine Länge von d_{min} oder d_{max} hat, muss man den minimalen und maximalen z-Wert im fragwürdigen (gelb markierten) Bereich ermitteln, was aber bei hardware-beschleunigtem Zeichnen nicht praktikabel ist. Deshalb wird der gelbe Bereich ignoriert bzw. gar nicht erst gezeichnet, und der neue Streifen gleich hinter dem ersten d_{min}-Streifen gerendert:

[6] Dies bezieht sich auf den Fall des Parallelblicks, wo $f > 0$ gilt. Im Falle des Kreuzblicks mit $f < 0$, kann man einfach d_{min} und d_{max} vertauschen.

[7] Hierbei ist noch anzumerken, dass dieser Streifen von einem Augen als flacher Streifen ohne Korrespondenzpunkte gesehen wird, so dass dieser Streifen unabhängig vom z-Buffer gezeichnet werden muss



...

Das Problem ist nun, dass man das zwar so mittels Framebuffer implementieren kann, aber nicht implementieren soll, da es bei OpenGL nicht erlaubt ist[3], gleichzeitig von einer Textur zu lesen und mittels Framebuffer auf sie zu zeichnen.

Die einfachste Lösung hierfür ist es, das SIRDS einfach doppelt in zwei Framebuffer zu zeichnen, und dabei jeweils von der Textur des anderen Buffers zu lesen:

Initialisierung mit Basistextur:

Schritt 1: Zeichnen in Buffer 1: (Pfeile bedeuten Blockkopieren, Striche mögliches Lesen von Pixeln)

Schritt 2: Kopieren und Zeichnen in Buffer 2:

Schritt 3: Kopieren und Zeichnen in Buffer 1:

...

Nach $\lceil \frac{width - d_{max}}{d_{min}} \rceil + 1$ Schritten hat man das komplette SIRDS in beiden Buffern.

Noch einfacher als das Kopieren und Rendern in zwei Buffern ist es, nie zu Kopieren, sondern in jedem Schritt (außer dem ersten) einen Streifen der Länge $2 \cdot d_{min}$ zu zeichnen. Dann rendert der SIRDS-Shader in die erste Hälfte des Streifens genau die Pixel, die kopiert werden würden.

Alternativ könnte man das SIRDS auch direkt in den Frontbuffer rendern, und dann mit `glCopyTexImage2D` in die vom Shader verwendete Textur kopieren. Dies könnte etwas schneller sein, da das SIRDS dabei nur einmal berechnet wird, allerdings ist es weniger flexibler, da die SIRDS-Größe/Auflösung durch die Bildschirmgröße bestimmt wird, und man das fertige SIRDS nicht herabskaliert ausgeben kann.

(siehe Abschnitt 6 für einen kurzen Vergleich der Varianten)

4 Animationen

Das Animieren eines SIRDS stellt ein großes Problem dar, da sich ein SIRDS nicht lokal ändern lässt, und jede Änderung eines Pixels auch dutzende weitere Pixel in späteren Streifen beeinflusst. Da das Auge sehr sensibel auf jegliche Änderungen reagiert, erscheinen diese beeinflussten Pixel hervorgehoben und man sieht mehrere ≫Schatten≪ des bewegten Objektes.

Um also ein SIRDS zu animieren, muss man das Gehirn des Betrachters daran hindern, den neu gerenderten Frame mit dem alten Frame zu vergleichen. Dafür scheint es nur zwei Techniken zu geben, entweder löscht man den Frame aus dem Gedächtnis der Netzhaut, oder man ändert den neuen Frame so stark, dass er nicht mehr mit dem alten vergleichbar ist.

Das Löschen lässt sich leicht dadurch erreichen, dass man einen leeren Frame ohne SIRDS rendert, so dass ein anschließend angezeigtes SIRDS als völlig neues Bild erscheint. Dies führt allerdings zu einem ziemlich unangenehmem Flackern, vor allem da der leere Zwischenframe auch mindestens ca. $\frac{1}{25}$ Sekunde lang angezeigt werden muss, um überhaupt richtig wahrgenommen zu werden. Dadurch ergibt sich eine maximale Framerate von 40 Bildern pro Sekunde (von denen nur 20 ein SIRDS und die eigentliche Szene zeigen), was zu ziemlich stockenden Bewegungen führt.

Die Alternative der starken Abänderung lässt sich einfach dadurch realisieren, dass man in dem neuen Frame eine andere Grundtextur benutzt oder den Texturoffset verschiebt. Dadurch flackert das Bild zwar ebenfalls, aber dies stört bei hohen Frameraten weniger. Bei der Benutzung des Programms stellt man zudem fest, dass die Texturen nicht zu heterogen sein dürfen, da man sonst ebenfalls nichts mehr erkennt.

5 Implementation

Das SIRDS-Programm basiert auf dem Framework von der CG-Seite.

Die Vorgehensweise des Programmes beim Zeichnen eines SIRDS ist Folgende:

1. Die Szene wird in einen Framebuffer mit angehängter Depthtexture gerendert
2. Die relevanten Daten (d, f und die Textursampler) werden an den Shader übergeben
3. Wie im Abschnitt 3 beschrieben, werden abwechselnd Streifen in zwei zusätzlichen Framebuffern mit aktiviertem Shader gezeichnet
4. Die Textur eines der beiden Framebuffers wird auf den Bildschirm gezeichnet

Der Shader berechnet dabei einfach für jeden Pixel den Abstand $d_{cur} = d + z(x, y) \cdot f$ und gibt dann die Farbe des korrespondierenden Pixels an der Position $(x - d_{cur}, y) = (x - d - z(x, y)f, y)$ zurück.

Der Einfachheit halber geht der Shader davon aus, dass für die Depthtextur dieselben Koordinaten wie für die Textur mit den Farben verwendet werden, so dass ein Punkt (x, y) tatsächlich den z-Wert $z(x, y)$ darstellt.

Die Position des betrachteten Pixels im z-Buffer erfährt der Shader dabei über die Texturkoordinaten, so dass man im letzten Schritt das SIRDS auch verschoben zeichnen könnte.

Das Programm bietet über die Tastatur verschiedene Einstellungsmöglichkeiten, so kann man sich die z-Buffertextur ansehen, den Augenabstand ändern, die Größe der verwendeten Texturregion ändern, das SIRDS invertieren...

6 Auswertung

Das Programm rendert mit den Shader schöne SIRDS (siehe Bild 1 bis Bild 5).

Dadurch, dass die Texturwerte interpoliert werden, wird zudem ein Problem der SIRDS-Softwarerenderer (siehe Bild 3, meinen anderen Renderer und [2]) gelöst, bei denen der Abstand $d + fz$ als ganze Zahl nur wenige diskrete Werte annehmen kann. Dadurch gibt es nur wenige unterschiedliche z-Werte, und das ganze Bild erscheint aus Terrassen-ähnlichen Schichten aufgebaut.

Durch die Interpolation tritt dieser Effekt jedoch nicht auf, und es gibt weiche Übergänge zwischen allen z-Werten. (dafür ist das Bild allerdings leicht unscharf)

Die Geschwindigkeit ist auch akzeptabel, aber es gibt deutliche Unterschiede zwischen den im Abschnitt 3 erwähnten Verfahren:

Erwartungsgemäß ist das verbotene gleichzeitige Lesen und Rendern in dieselbe Textur am schnellsten. Auf meinem Computer erreicht es eine Framerate von ca. 110 FPS, was genauso schnell ist, wie das Rendern der Szene in einen Depthbuffers-Framebuffer mit anschließender Ausgabe dieses Buffers auf einem bildschirmfüllenden Polygon. (Die Ausgabe der Szene alleine erreicht 145 FPS)

Theoretisch sollte diese Variante auch ohne Probleme laufen, da dabei in einem Schritt von keinem Pixel gelesen wird, dessen Wert tatsächlich verändert wurde. Trotzdem widerspricht es der OpenGL-Spezifikation, und es könnte Treiber geben, bei denen es nicht funktioniert (die z.B.: sicherheitshalber immer nur schwarz zurückgeben, wenn die Textur eventuell verändert wurde), weshalb man es nicht als einziges Verfahren implementieren sollte.

Das andere Verfahren mit zwei Framebuffern erreicht 55 FPS. Vermutlich liegt dies an dem doppelten Zeichnen bildschirmfüllender Polygone und dem

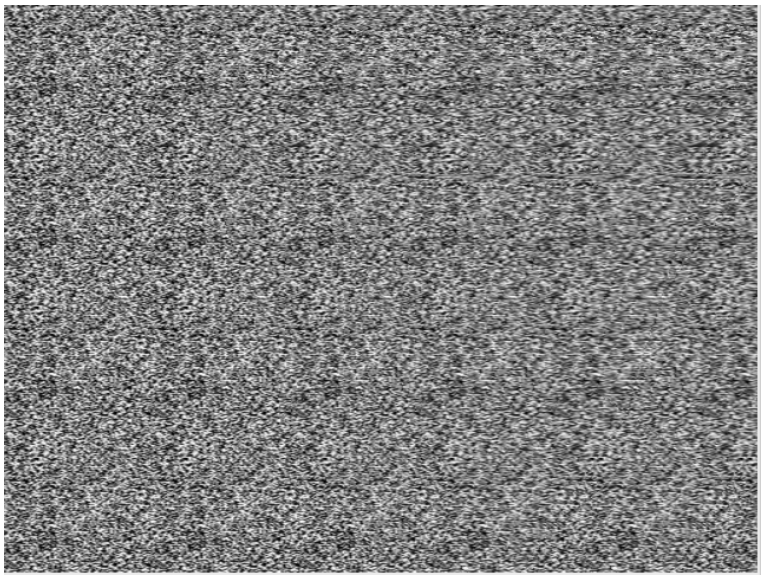


Abbildung 1: Ein Würfel

ständigen Wechseln zwischen den Framebuffern. Es liegt jedoch nicht am SIRDS-Shader, denn wenn man die Streifen ohne Shader und nur mit Textur rendert, ergibt sich die gleiche Framerate. Zeichnet man alles in denselben Framebuffer, ohne zu wechseln, ergibt sich eine Framerate von 75 FPS (aber man erhält natürlich kein SIRDS)

Literatur

[1] Wikipedia - stereoskopie. `http://de.wikipedia.org/wiki/Stereoskopie`.

[2] Lewey Geselowitz. The AbSIRD Project: To Create Real-Time SIRDs. `http://www.leweyg.com/download/SIRD/AbSIRD/essay.html`.

[3] Kurt Akeley Mark Segal. *The OpenGL Graphics System: A Specification*. The Khronos Group Inc., 3.0 edition, August 2008.

Abbildung 2: Ein Hase

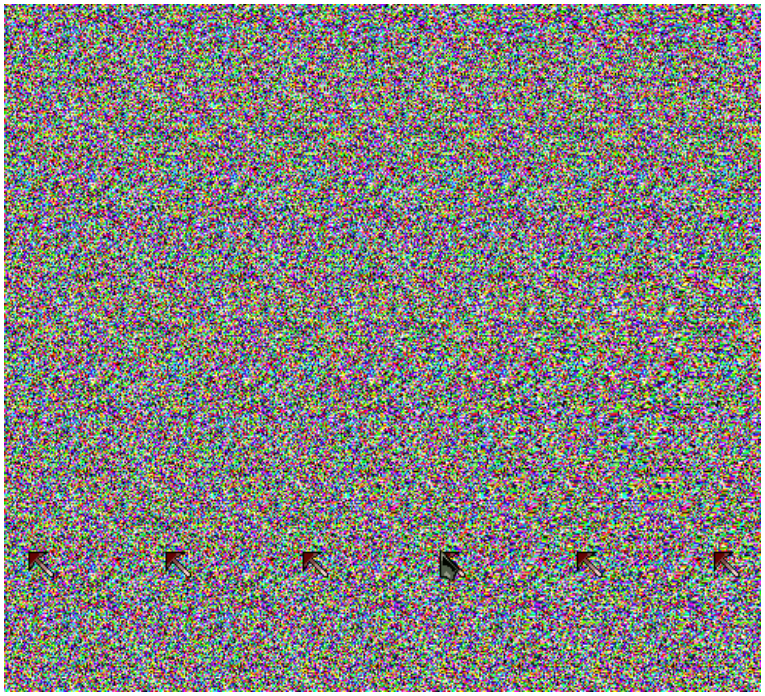


Abbildung 3: Ein Würfel ohne Texturinterpolation mit Floatercursor

Abbildung 4: Ein invertierter Hase

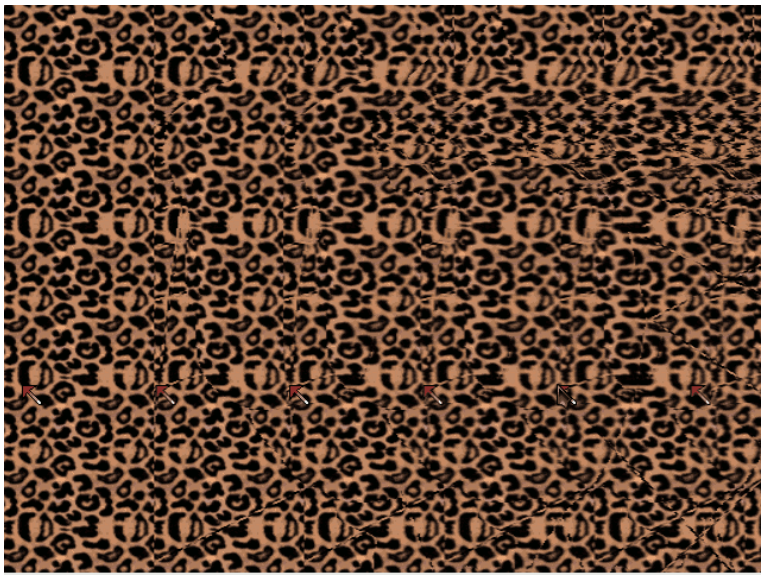


Abbildung 5: Ein Hase (SIS) mit Floatercursor