VEKTORIA-MANUAL
FRAMES & VIEWPORTS

Prof. Dr. Tobias Breiner

vektoria

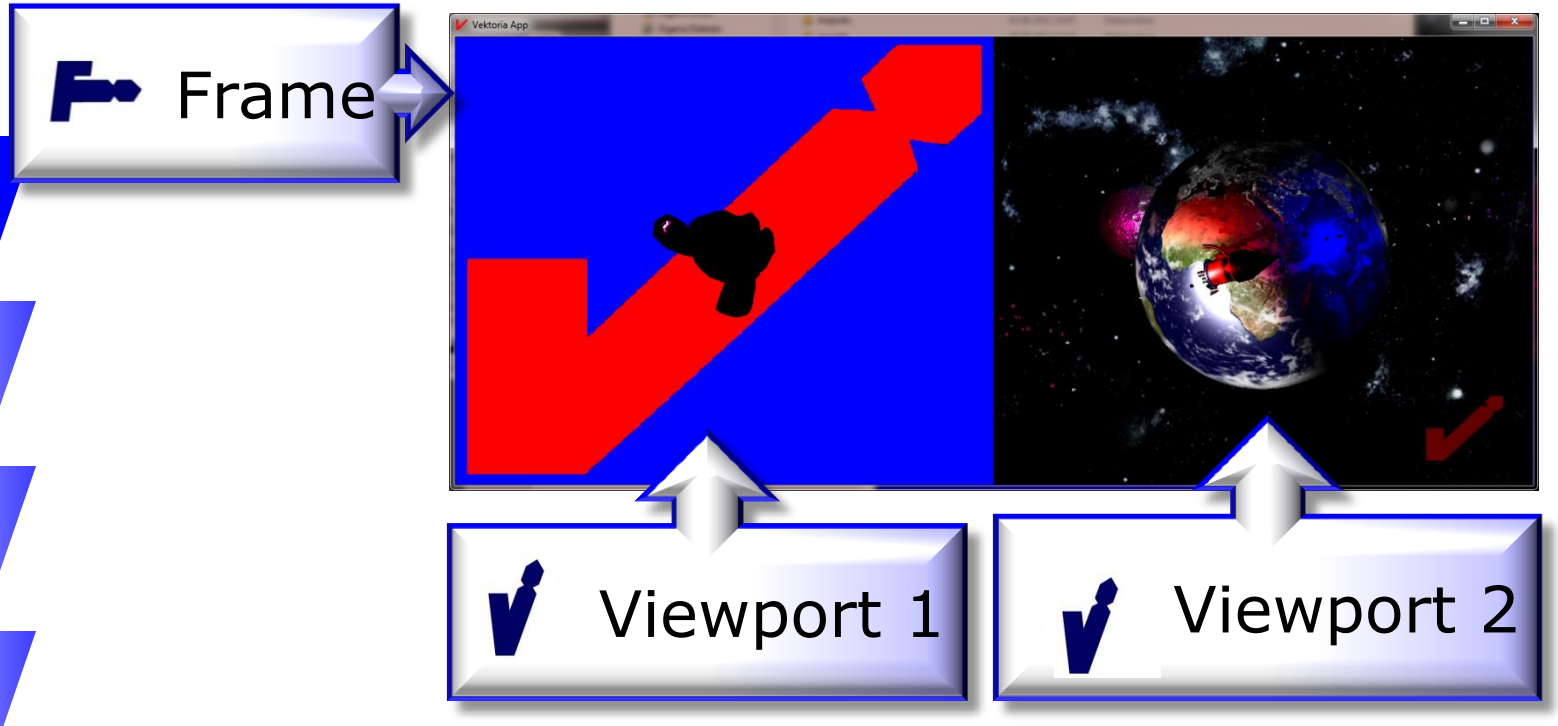
www.vektoria-engine.com

Inhalt

Kapitel 1

Übersicht über Frames und Viewports

Frames & Viewports - Beispiel



Kapitel 2

Was ist ein Frame?

- Ein Frame ist ein rechteckiger Renderbereich, der in einem Fensterrahmen (Window) läuft.
- Ein Frame hat eine Render-Api (z.B. Direct3D, OpenGL oder NullRenderer) und eine Input-Api (z.B. DirectInput, STL oder Null_Device).
- Auf einem Computer können bis zu vier Frames laufen (mehr macht sowieso keinen Sinn).
- Jeder Frame kann beliebig viele Viewports (Sichtfenster) und Devices (Ein- und Ausgabegeräte wie Maus, Tastatur, Game Controller) enthalten.

CFrame::Init

```
CRenderApi* Init(HWND hwnd,
  EApiRender eApiRender
       = eApiRender_DirectX11_Shadermodel50,
  EApiInput eApiInput
       = eApiInput_DirectInput,
  EApiSound eApiSound
       = eApiSound_DirectSound,
  EShaderCreation eShaderCreation
       = eShaderCreation_CompileChanges,
  EShaderAutoRecompilation eShaderAutoRecompilation
       = eShaderAutoRecompilation_Enabled);
```

Initialisiert den Frame und gibt einen Zeiger auf die automatisch kreierte RenderApi zurück. Der Methode muss als erten Parameter zwingend ein ein Handle auf das Windowsfenster, in das CFrame hineinmalt übergeben werden (hwnd). Dazu existieren fünf optionale Parameter, die auf den nächsten Seiten detailiert erklärt werden: eApiRender, eApiInput, eApiSound, eShaderCreation und eShaderAutoRecompilation.

CFrame::Init (eApiRender)

eApiRender bestimmt die gewünschte Grafik-Basis-Api.
Zurzeit sind folgende Eingabewerte möglich:

- eApiRender_DirectX11_Shadermodel41
  abgespeckter DirectX11-Renderer mit Shader Modell 4.1, diese Option ist sinnvoll, wenn Sie einen älteren Rechner (um 2009) besitzen, der nicht Shadermodell 5.0 unterstützt.
- eApiRender_DirectX11_Shadermodel50
  DirectX11-Renderer mit Shader Modell 5.0, dies ist die Default-Einstellung
- eApiRender_DirectX11_ForwardPlus
  bester Renderer mit ForwardPlus-Technologie, basiert ebenfalls auf DirectX11 mit dem Shader Modell 5.0 allerdings ist die Parameterjustierung bei Lichtern komplex.
- eApiRender_Null
  Null-Renderer, bei dem die Renderanweisungen ins Leere laufen, ist nur sinnvoll zum Debuggen.

CFrame::Init (eApiSound)

eApiSound bestimmt die gewünschte Grafik-Basis-Api.
Zurzeit sind folgende Eingabewerte möglich:

- eApiSound_DirectSound
  Es wird für Musik und Klänge und Geräusche die DirectSound-Api geladen (Default)
- eApiSound_DirectAudio
  Es wird für Musik und Klänge und Geräusche die DirectAudio-Api geladen
- eApiSound_OpenAL
  Es wird für Musik und Klänge und Geräusche die OpenAL-Api geladen
- eApiSound_Null
  Null-Sound-Api, bei dem die Soundanweisungen ins Leere laufen, ist nur sinnvoll zum Debuggen.

CFrame::Init-Parameter

eApiInput gibt die gewünschte Eingabe-Api an.
Zurzeit sind zurzeit folgende Optionen erlaubt:

- eApiInput_DirectInput (Default)
  Die Eingabe-Api basiert auf DirectInput.
- eApiInput_Null
  Die Eingabe-Abfragen laufen hier ins Leere.
  Nur Sinnvoll zum Debuggen.

CFrame::Init-Parameter

eShaderCreation gibt die Art der Shadererzeugung an.
Es sind drei Optionen möglich:

- eShaderCreation_ForceCompile
  Hier werden alle Shader zwingend am Anfang neu kompiliert. Damit startet das Programm aber wesentlich später.
- eShaderCreation_CompileChanges
  Hier werden nur diejenigen Shader kompiliert, die sich seit dem letzten Mal geändert haben. Dies ist die Default-Option
- eShaderCreation_UseCached
  Hier wird kein Shader übersetzt, sondern nur die schon übersetzten Shader aus dem Cache verwendet. Veränderungen am Shader-Source-Code wirken sich daher nicht auf die Applikation aus.

CFrame::Init-Parameter

eShaderAutoRecompilation gibt den Zeitpunkt der Shader-Kompilierung an.
Wirkt sich naturgemäß nicht aus, falls der vorherige Parameter eShaderCreation auf eShaderCreation_UseCached gesetzt wurde.

Es sind folgende Optionen möglich:

- eShaderAutoRecompilation_Disabled
  Es werden die Shader nur beim Programmstart kompiliert.
- eShaderAutoRecompilation_Enabled
  Hier werden Shader automatisch rekompiliert, wenn sie verändert wurden, auch währen der Laufzeit. Dies ist der Default-Wert.

Kapitel 3

1

2

VIEWPORTS

4

5

Viewport-Initialisierung

Will man einen Viewport erzeugen, der den ganzen Frame ausfüllt (Normalfall), dann verwendet man folgende Initalisierungsmethode:

```
void InitFull(CCamera * pcamera);
```

Hier wird ein Pointer auf die Kamera übergegeben, deren Sicht das Viewport anzeigen soll.

Viewport-Initialisierung

Manchmal soll ein Viewport nur einen Teil des Frames ausfüllen, z.B. für:

- Split-Screens
- In-Screens
- Dreitafelprojektionen
- Triptychon-Darstellung
- etc.

Dafür verwendet man die Initialisierungsfunktion Init:

```
void Init(CCamera * pcamera,
      float frx, float fry, float frWidth,
      float frHeight);
```

Viewports-Initialisierung

Die Größe eines Viewports bezieht sich immer auf den übergeordneten Frame.
Die Größe wird bei der Initialisierung fraktional angegeben.
Dabei ist die linke obere Ecke gleich (x=0.0,y=0.0) und die rechte untere Ecke gleich (x=1.0,y=1.0)

Beispiel:

So wird ein Spilt-Screen erzeugt:

```
m_rvLeft.Init(&m_zcLeft,0.0f,0.0f,0.5f,1.0f);
m_rvRight.Init(&m_zcRight,0.5f,0.0f,0.5f,1.0f);
```

Sehr praktisch: Wird ein Frame vergrößert oder verkleinert ändern sich automatisch alle seine Viewports sinnfällig mit.

Viewport-Ansichten

Für ein Viewport können verschiedene Rendering-Parameter ein- und ausgeschaltet werden.

Sie können auch in einem bestimmten Stil gerendert werden.

Kapitel 4

Viewport-Schaltmethoden

1. `void SetBackfaceCullingOn();`
   `void SetBackfaceCullingOff();`

   Schaltet das Backface-Culling für den Viewport an oder aus (Default = an)

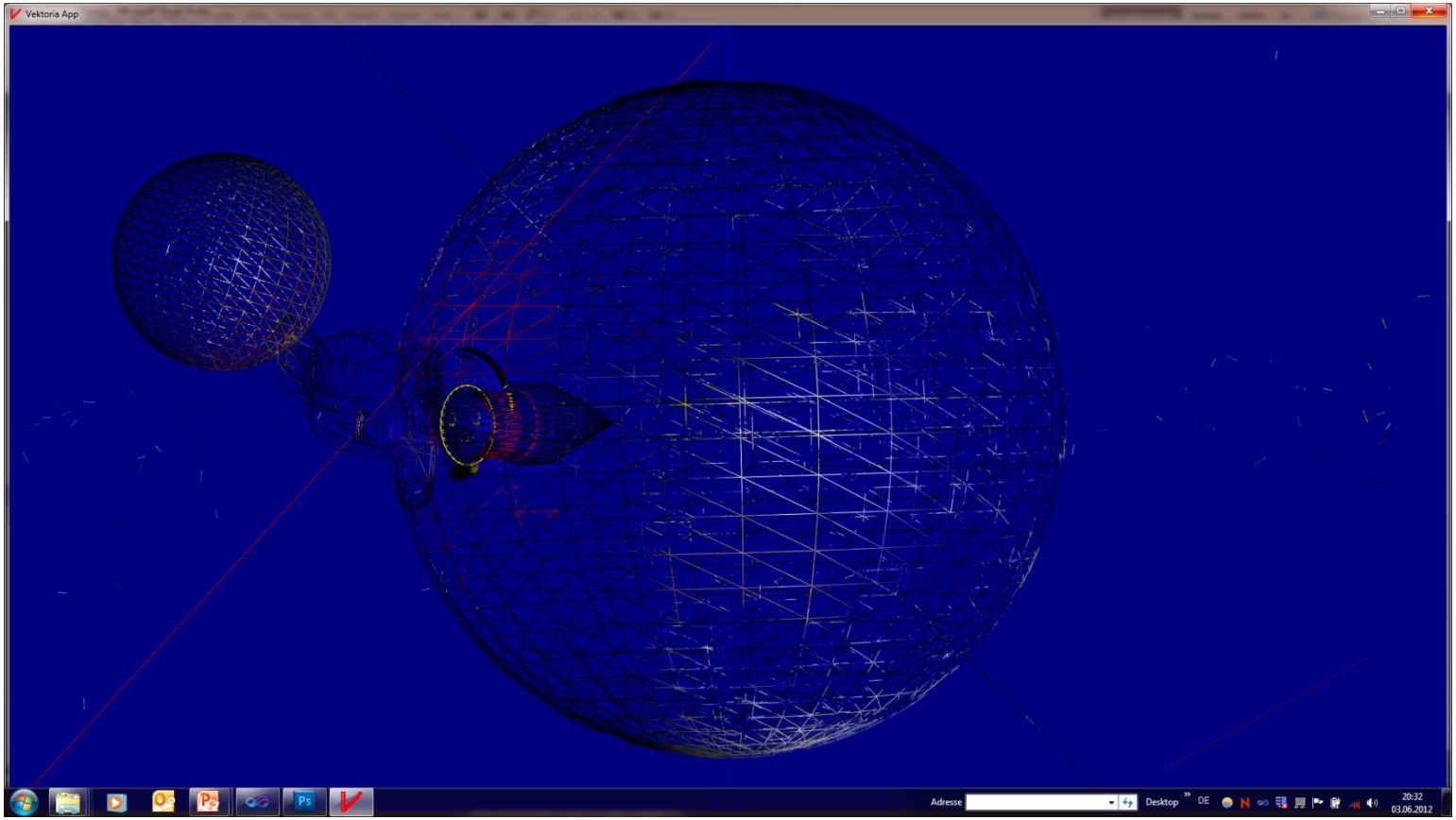
2. `void SetAntialiasingOn();`
3. `void SetAntialiasingOff();`

   Schaltet Antialiasing für den Viewport an oder aus (Default = an), funktioniert natürlich nur, wenn die Grafikkarte dies unterstützt.

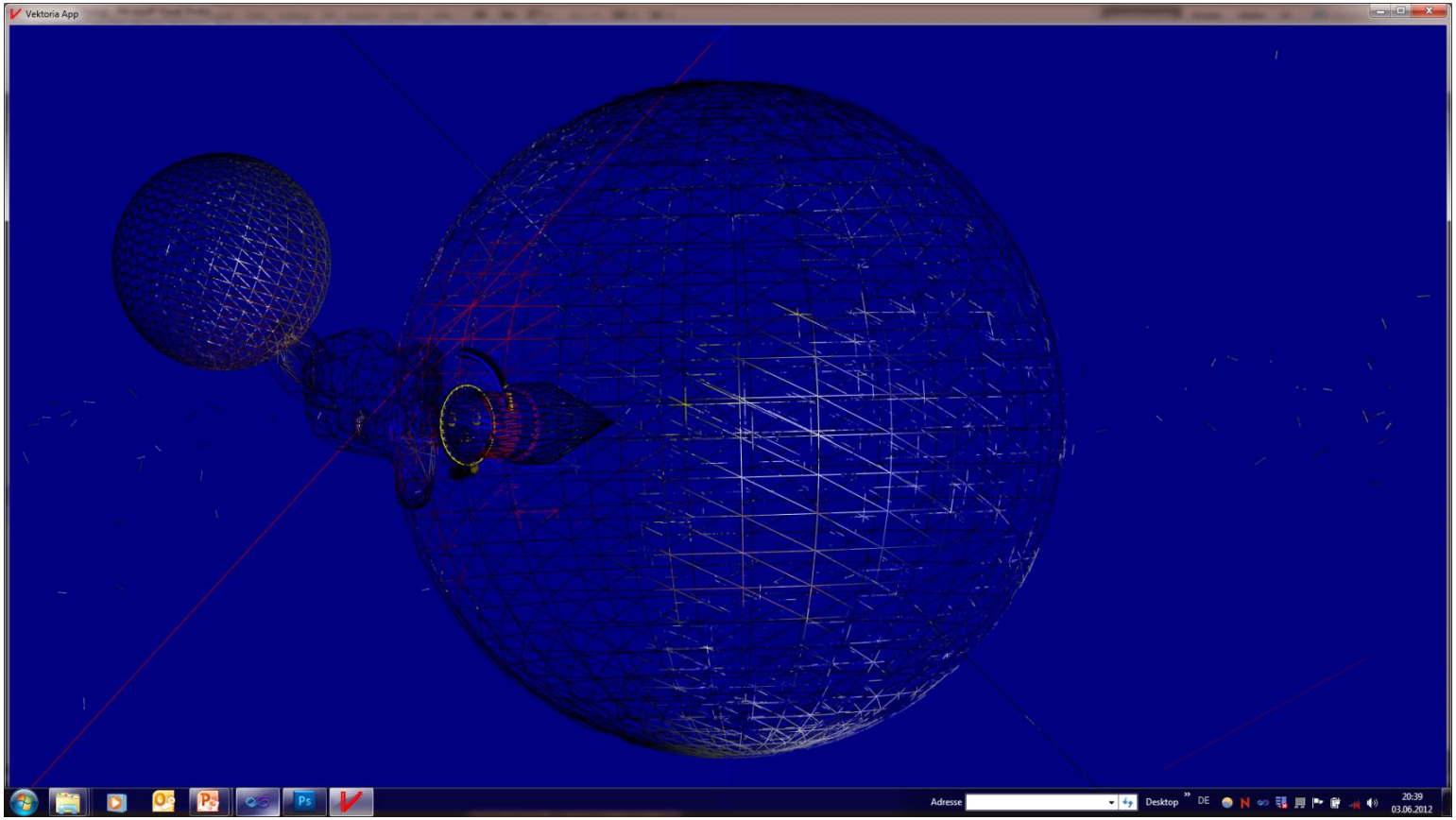
4. `void SetWireframeOn();`
5. `void SetWireframeOff();`

   Schaltet Drahtgittermodellansicht an oder aus (Default = aus).

SetWireframeOn



WireframeOn+BackfaceCullingOff



Weitere Viewport-Schaltmethoden

1
```
void SetShadowRenderingOn();
// Schaltet Schatten für den Viewport an (Default: an)
void SetShadowRenderingOff();
// Schaltet Schatten für den Viewport aus (Default: an)
```
2
```
void SetPointLightRenderingOn();
// Schaltet Punktlichter für den Viewport an (Default: an)
void SetPointLightRenderingOff();
// Schaltet Punktlichter für den Viewport aus (Default: an)
```
3
```
void SetParallelLightRenderingOn();
// Schaltet Parallellichter für den Viewport an (Default: an)
```
4
```
void SetParallelLightRenderingOff();
// Schaltet Parallellichter für den Viewport aus (Default: an)
void SetSpotLightRenderingOn();
// Schaltet Schweinwerferlichter für den Viewport an (Default: an)
```
5
```
void SetSpotLightRenderingOff();
// Schaltet Scheinwerferlichter für den Viewport aus (Default: an)
```

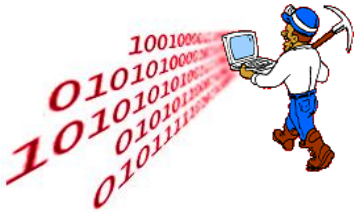
Weitere Viewport-Schaltmethoden

```
void SetShadowRenderingOn();
void SetShadowRenderingOff();
void SetPointLightRenderingOn();
void SetPointLightRenderingOff();
void SetParallelLightRenderingOn();
void SetParallelLightRenderingOff();
void SetSpotLightRenderingOn();
void SetSpotLightRenderingOff();
```

Weitere Schaltmethoden des Viewports, um Lichter und Schatten zu Anschauungszwecken aus- und anzuschalten!

Achtung, da aus technischen Gründen die obigen Schaltmethoden die jeweiligen Features nur zu Anschauungszwecken unterdrücken, wird keine Framerate-Beschleunigung erzielt!
Diese Schaltmethoden sollten daher nur in Ausnahmefällen verwendet werden! Will man kein Licht oder Schatten haben, einfach nicht implementieren!

Übung zu Viewports



1. Erzeugen Sie einen Split-Screen, wobei der linke Viewport die gleiche Kameraansicht rendern soll, wie die rechte, nur als Drahtgittermodell!

2.

3. Freiwillige Zusatzaufgabe für die schnellen Nerds:
   - Lassen Sie die Teilungslinie des Split-Screens während der Laufzeit variieren!

4.

5.

Kapitel 5

Stile

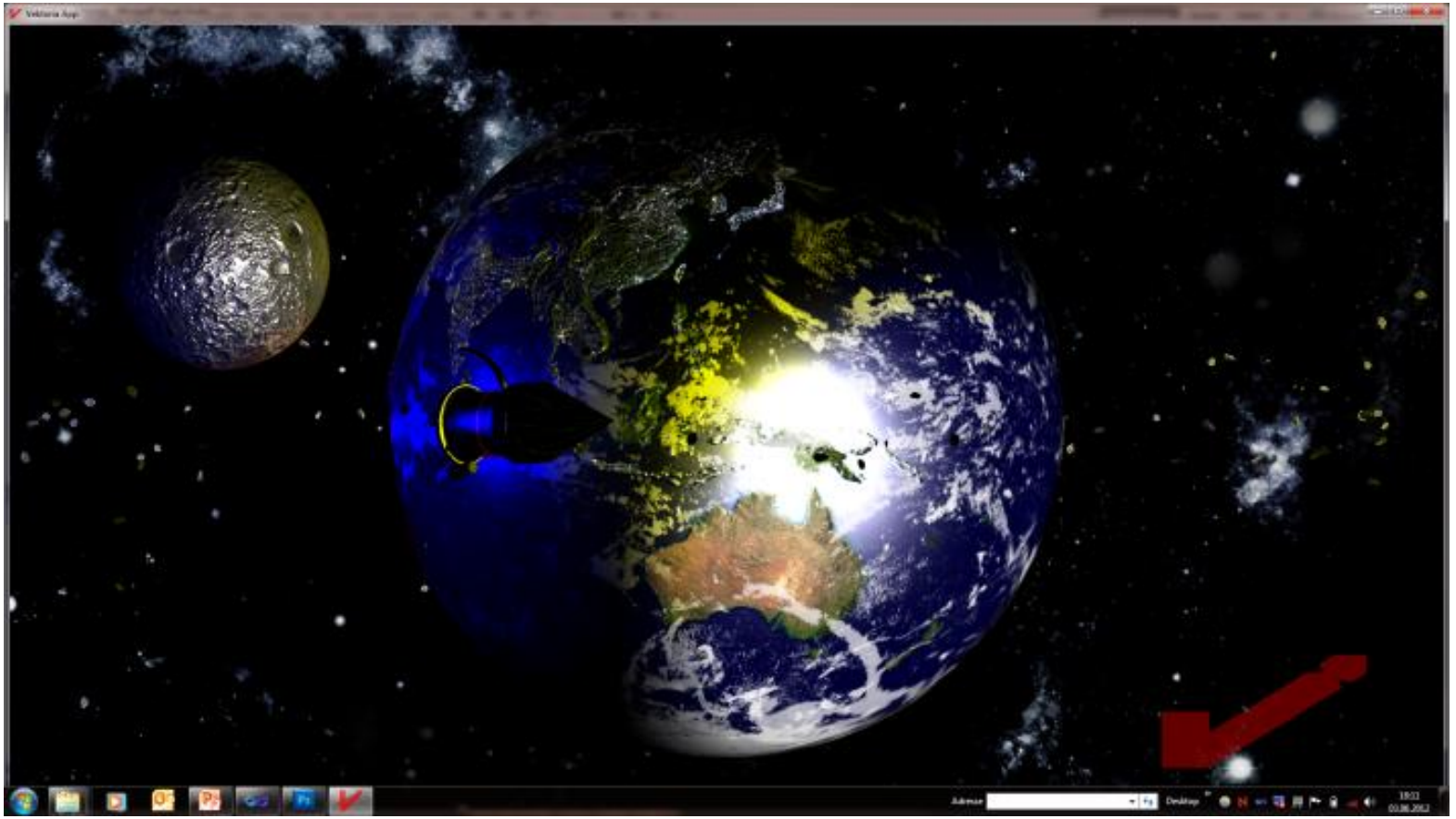
Ein Viewport kann in einem bestimmten Stil gerendert werden. Die betreffenden Methoden der Klasse CViewport fangen mit „Style“ an, z.B.:

```
m_zv.StyleSepia(); // der Viewport m_zv
                   // wird jetzt in Sepia
                   // gezeichnet
```

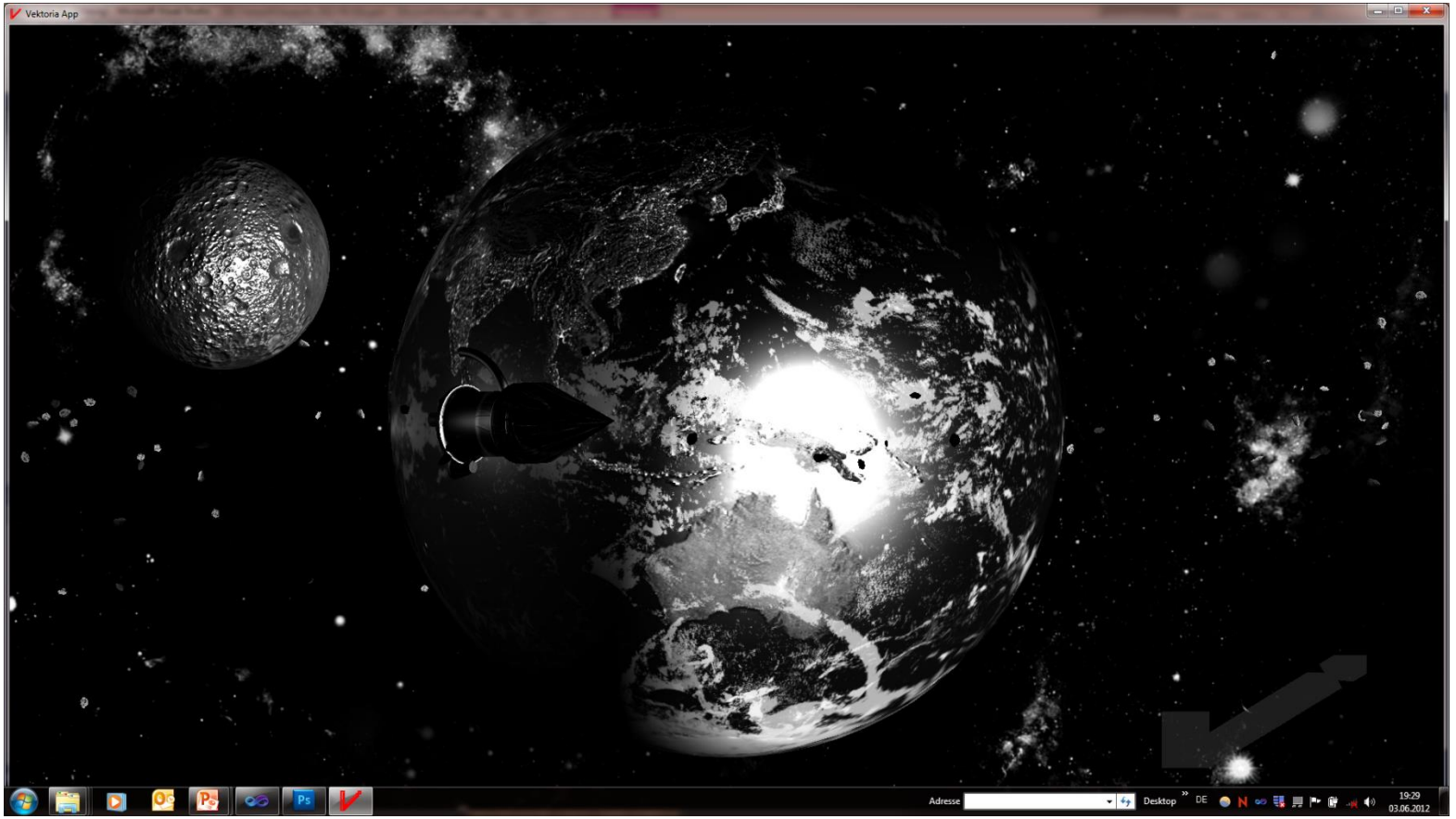
Stile und Schaltmethoden können beliebig miteinander kombiniert werden.

Stile können auch während der Laufzeit geändert werden.

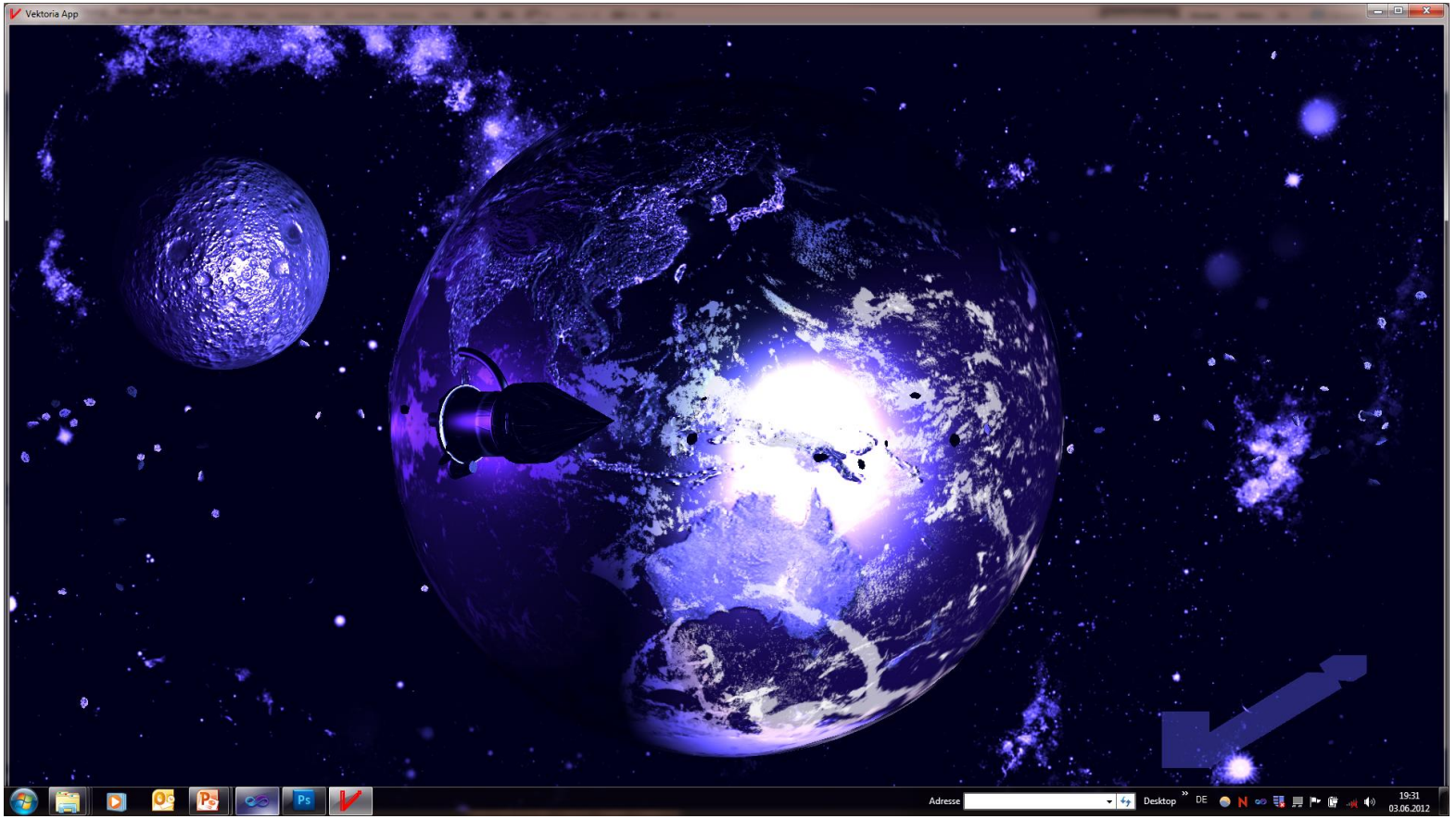
Ohne Style



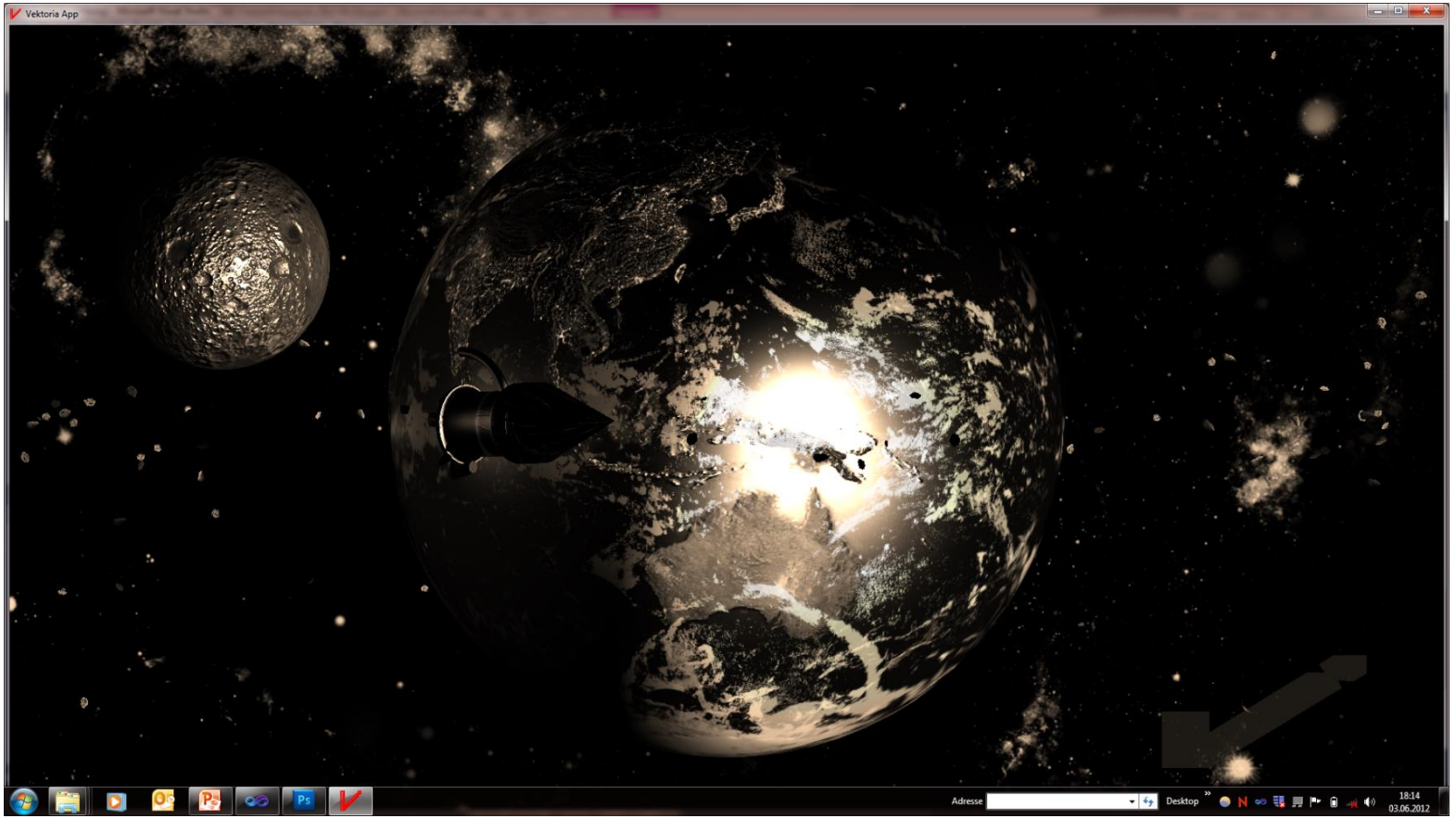
StyleBW



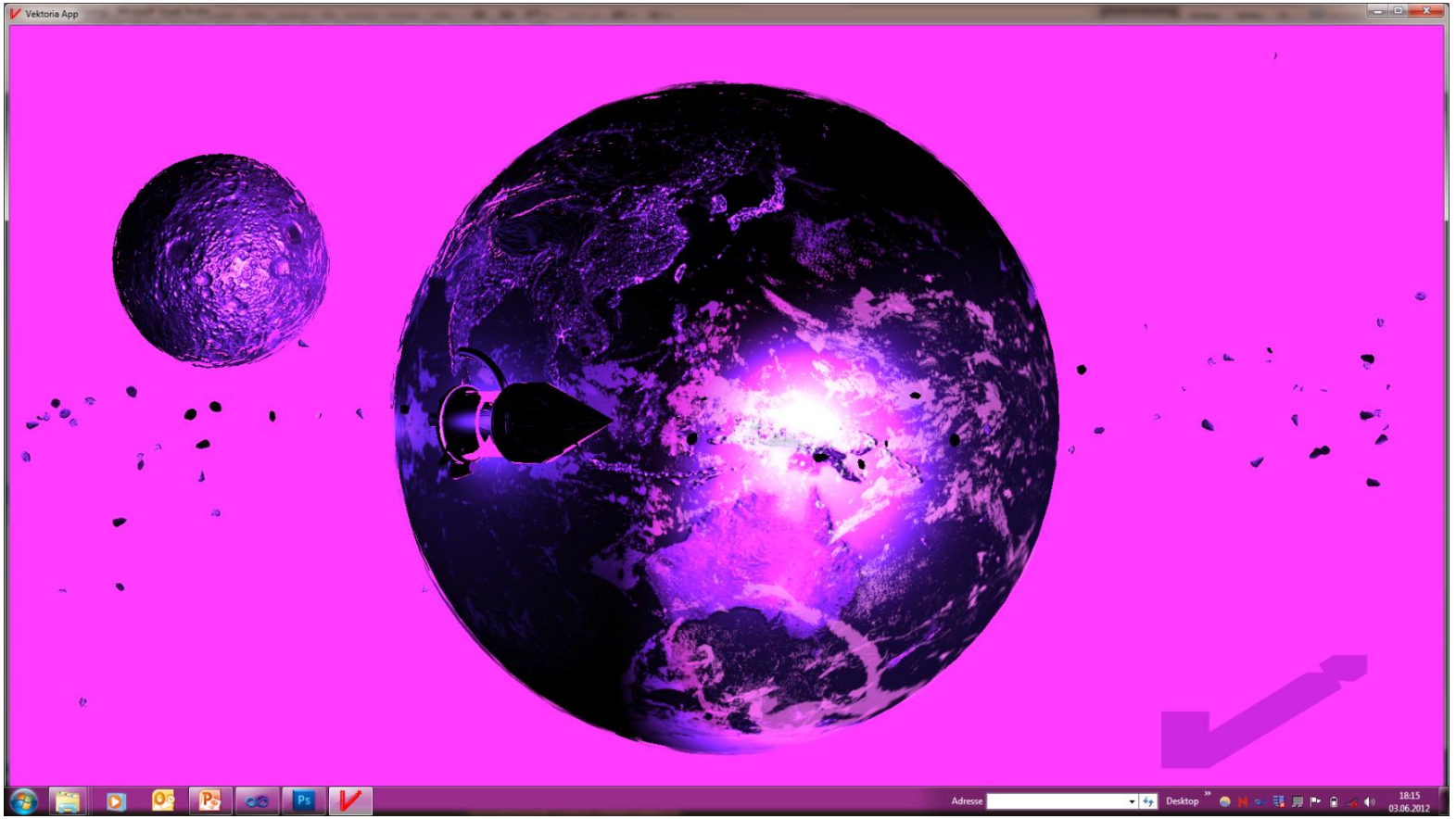
StyleBlueDream



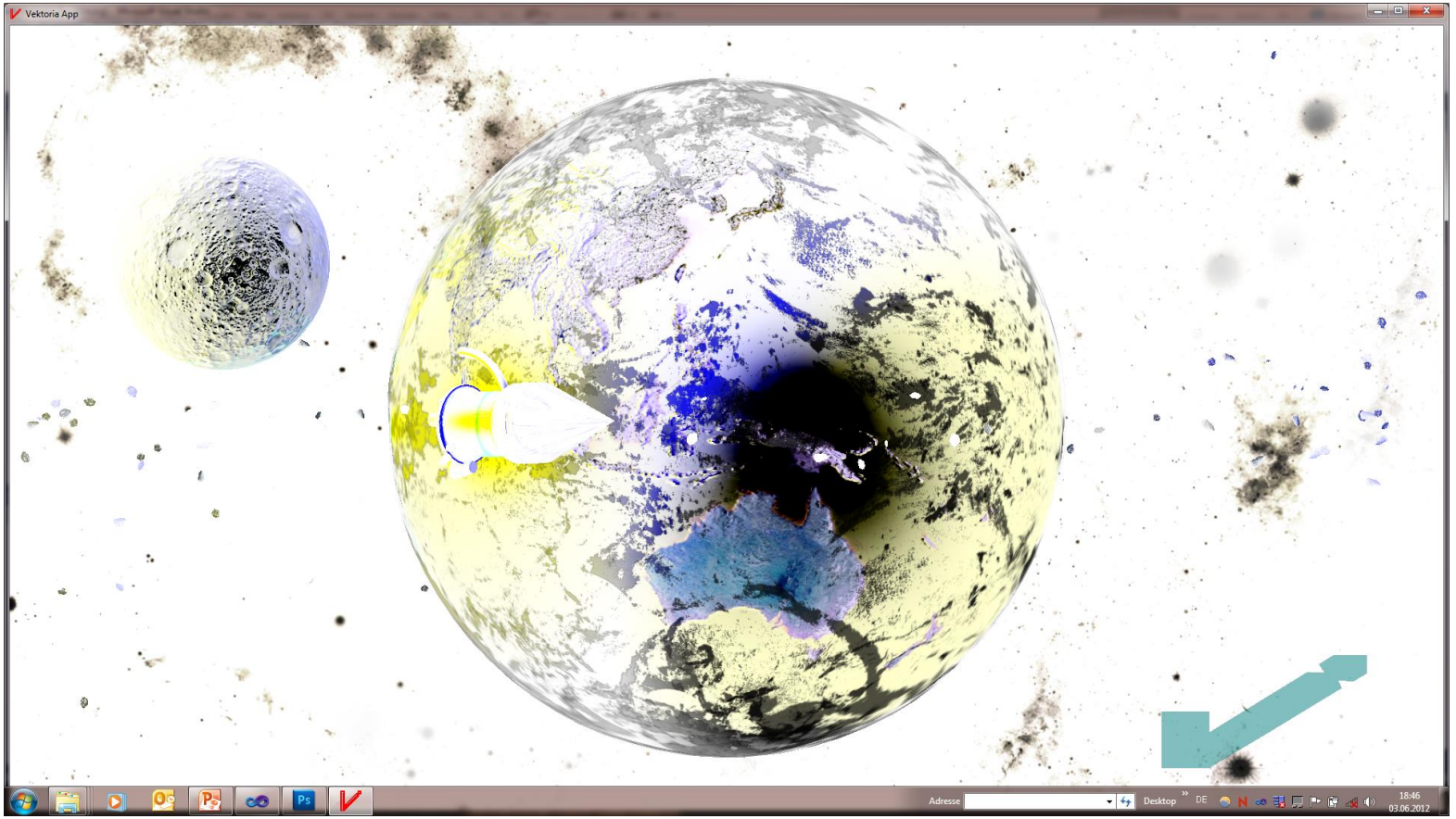
StyleSepia



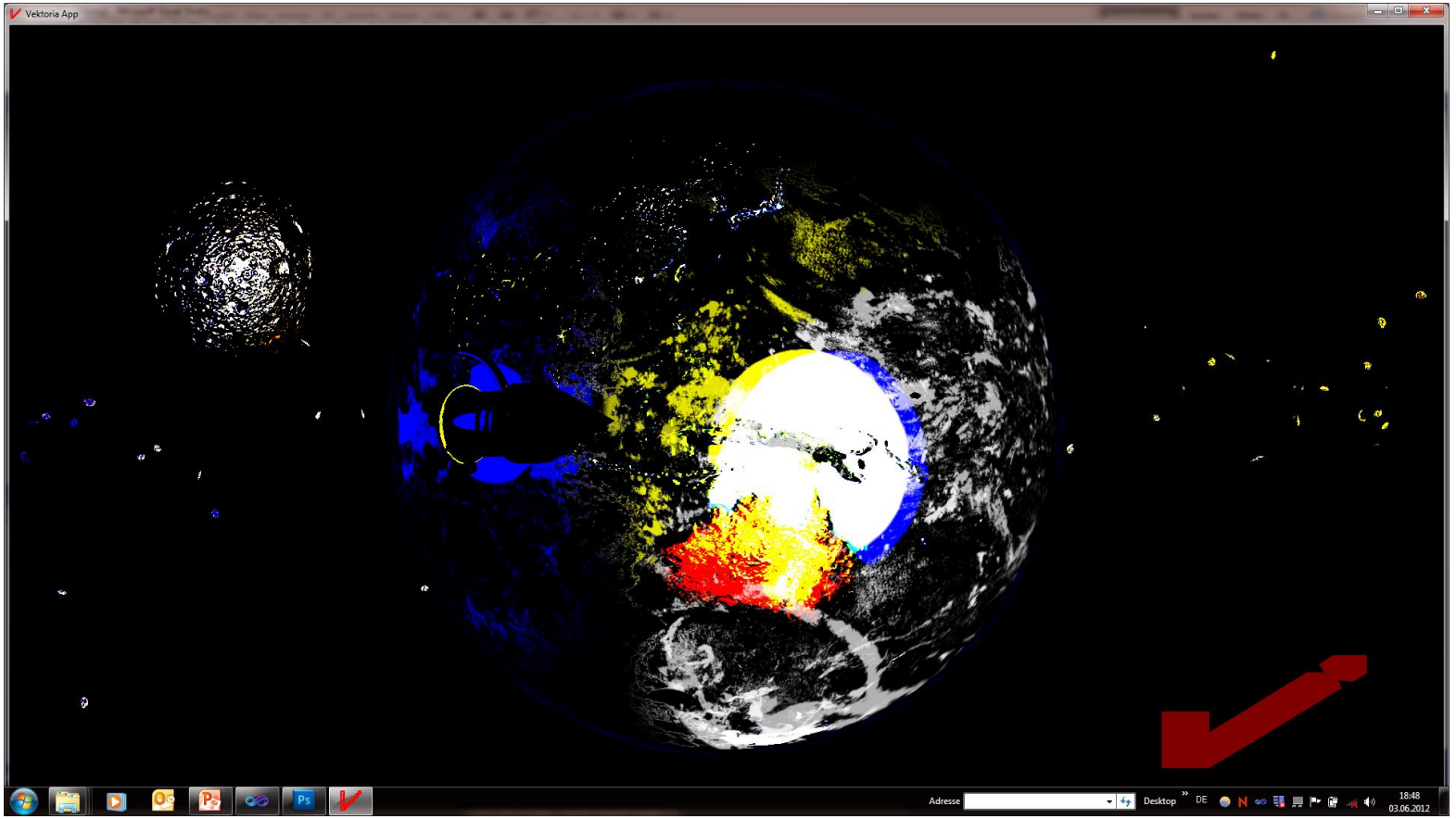
StylePurpleHaze



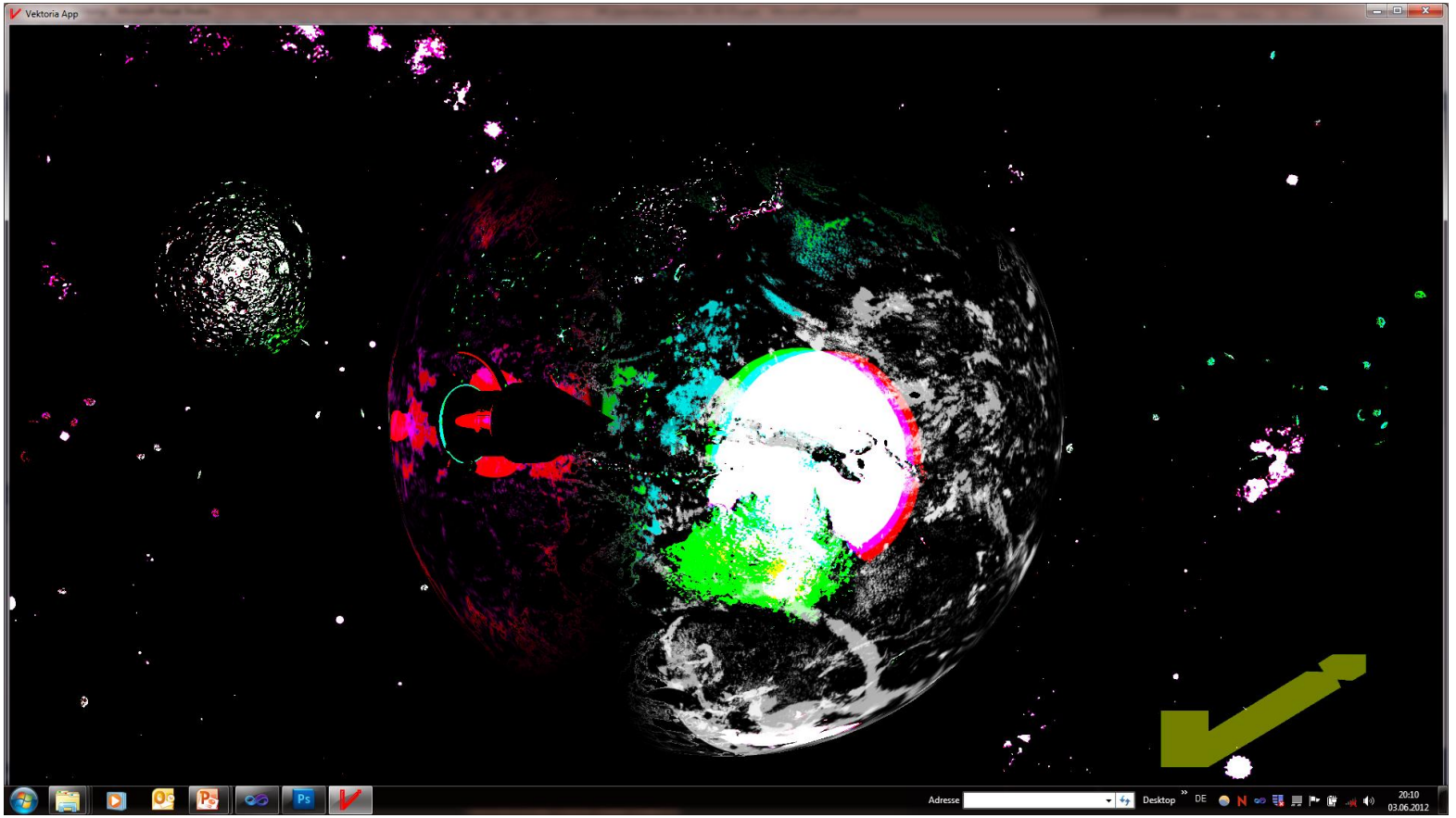
StyleInverse



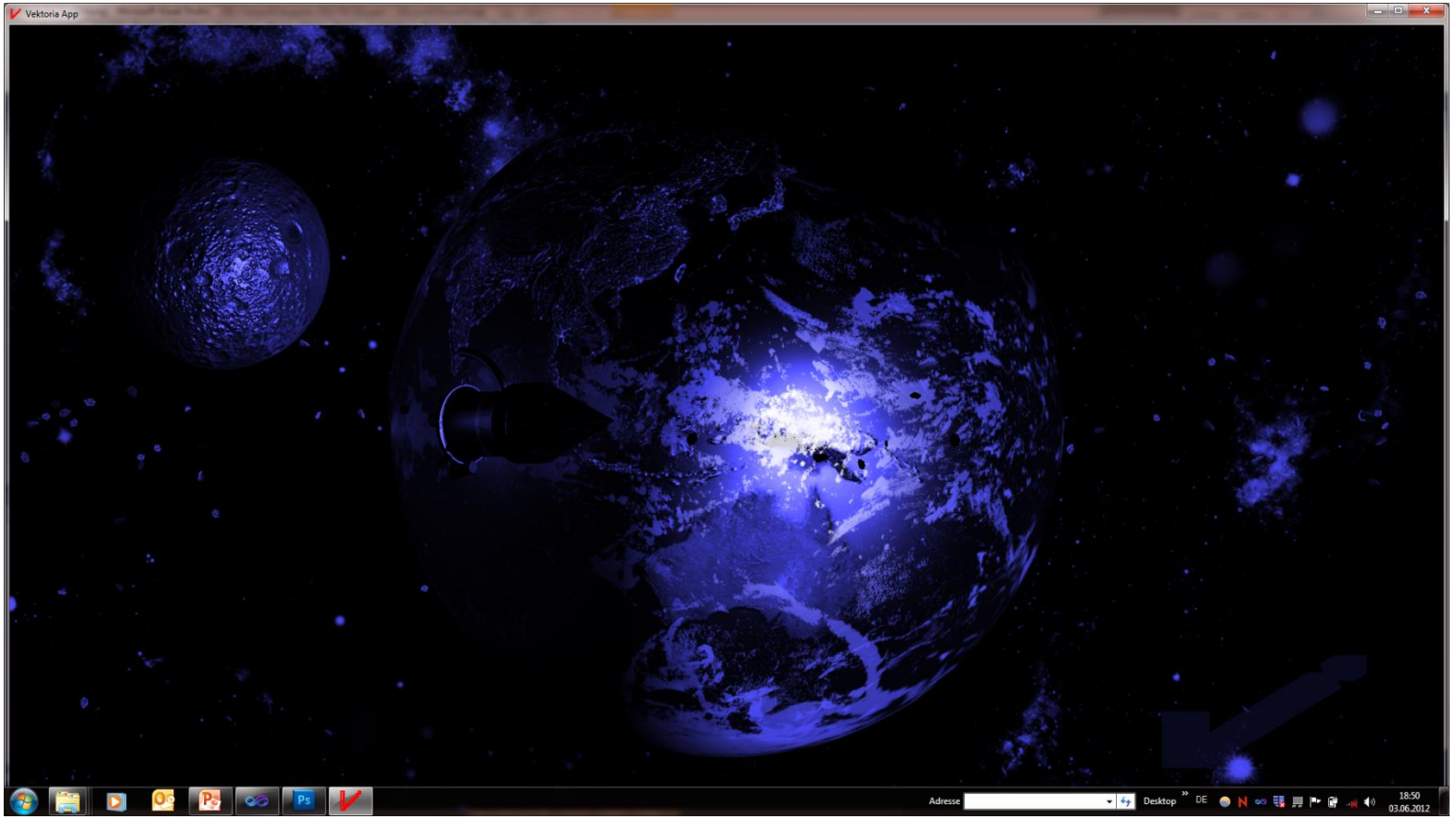
StyleFloppyPoppy



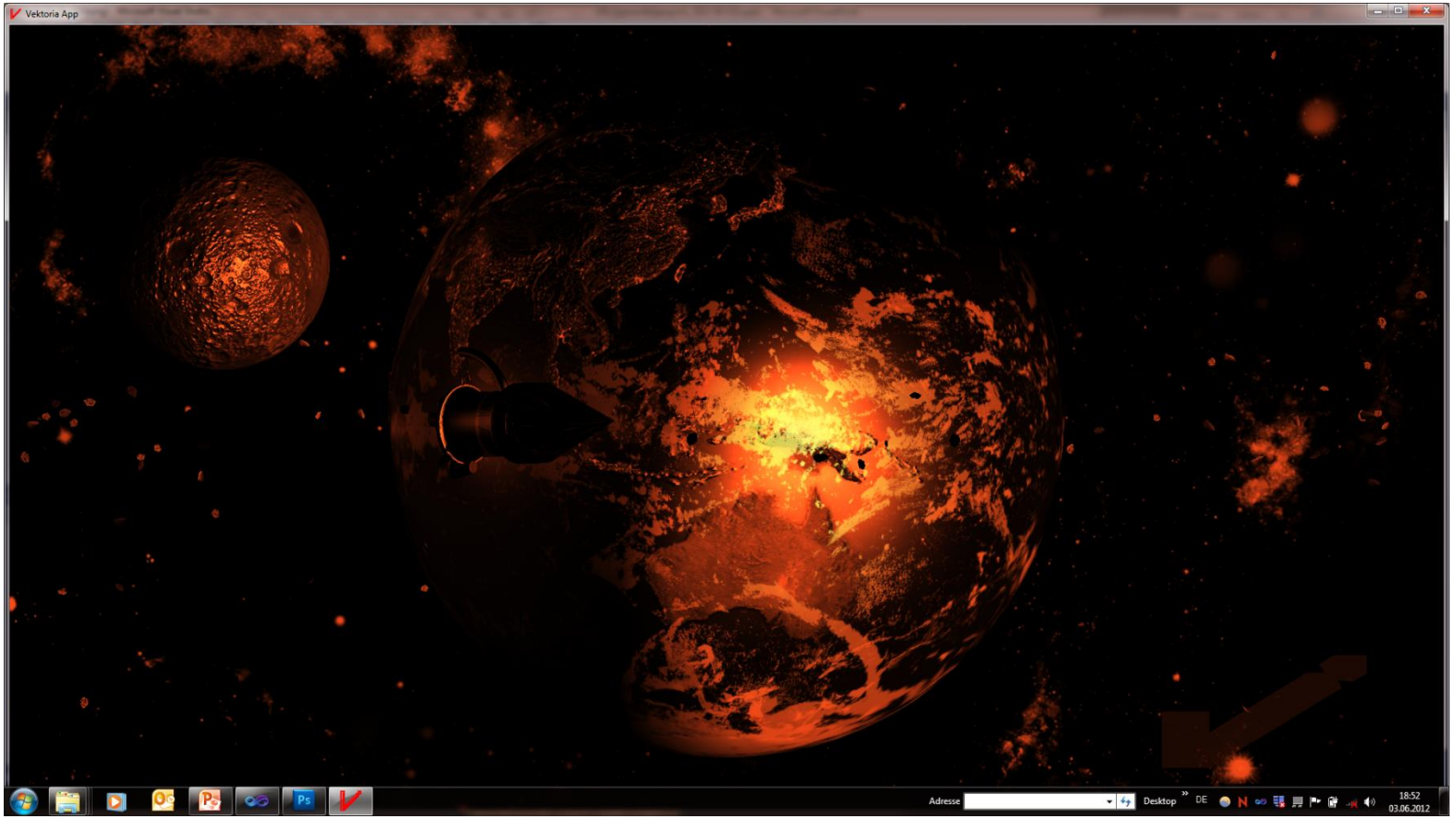
StylePopArt



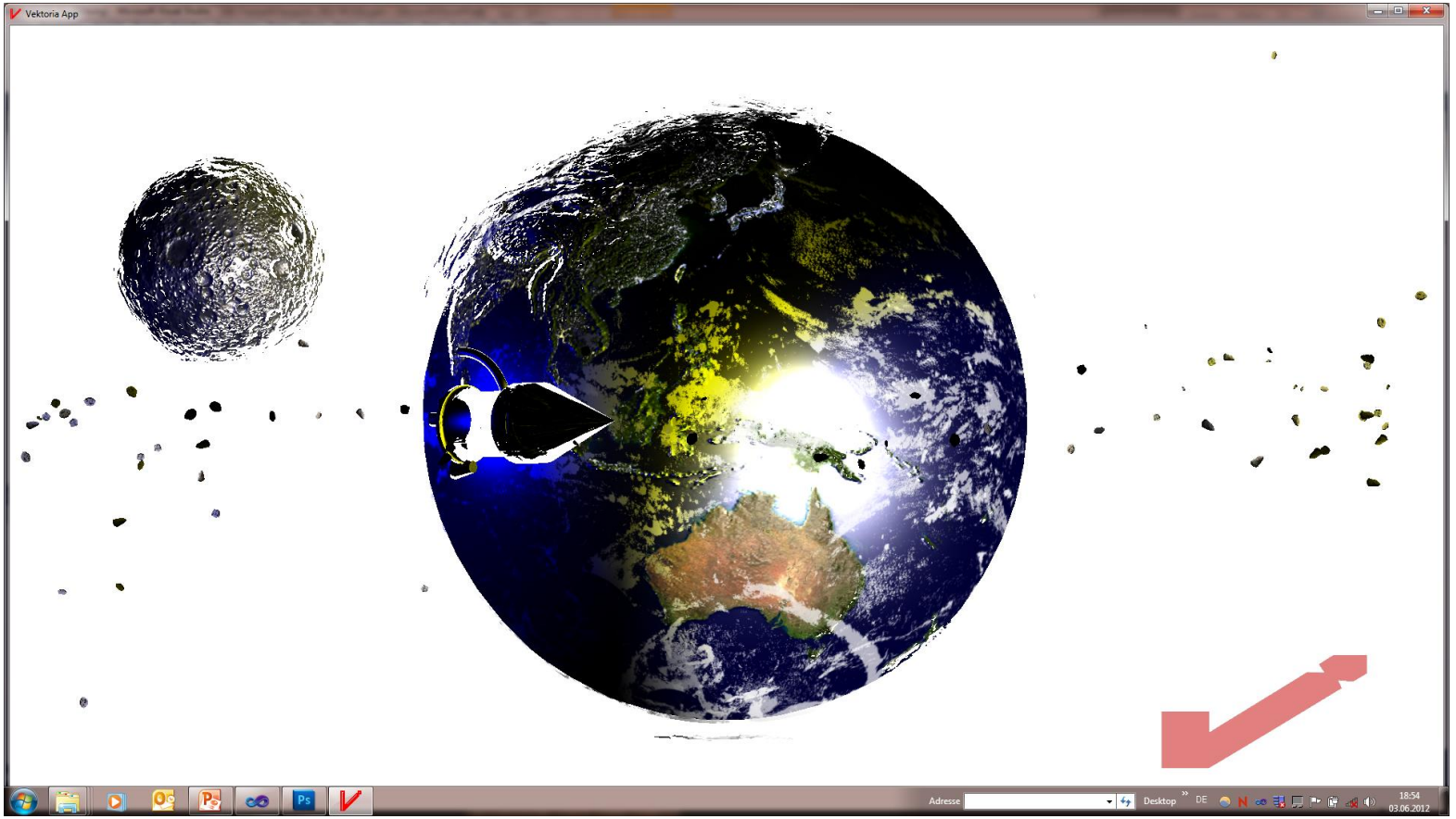
StyleMonochrome (0.3,0.3,1.0)



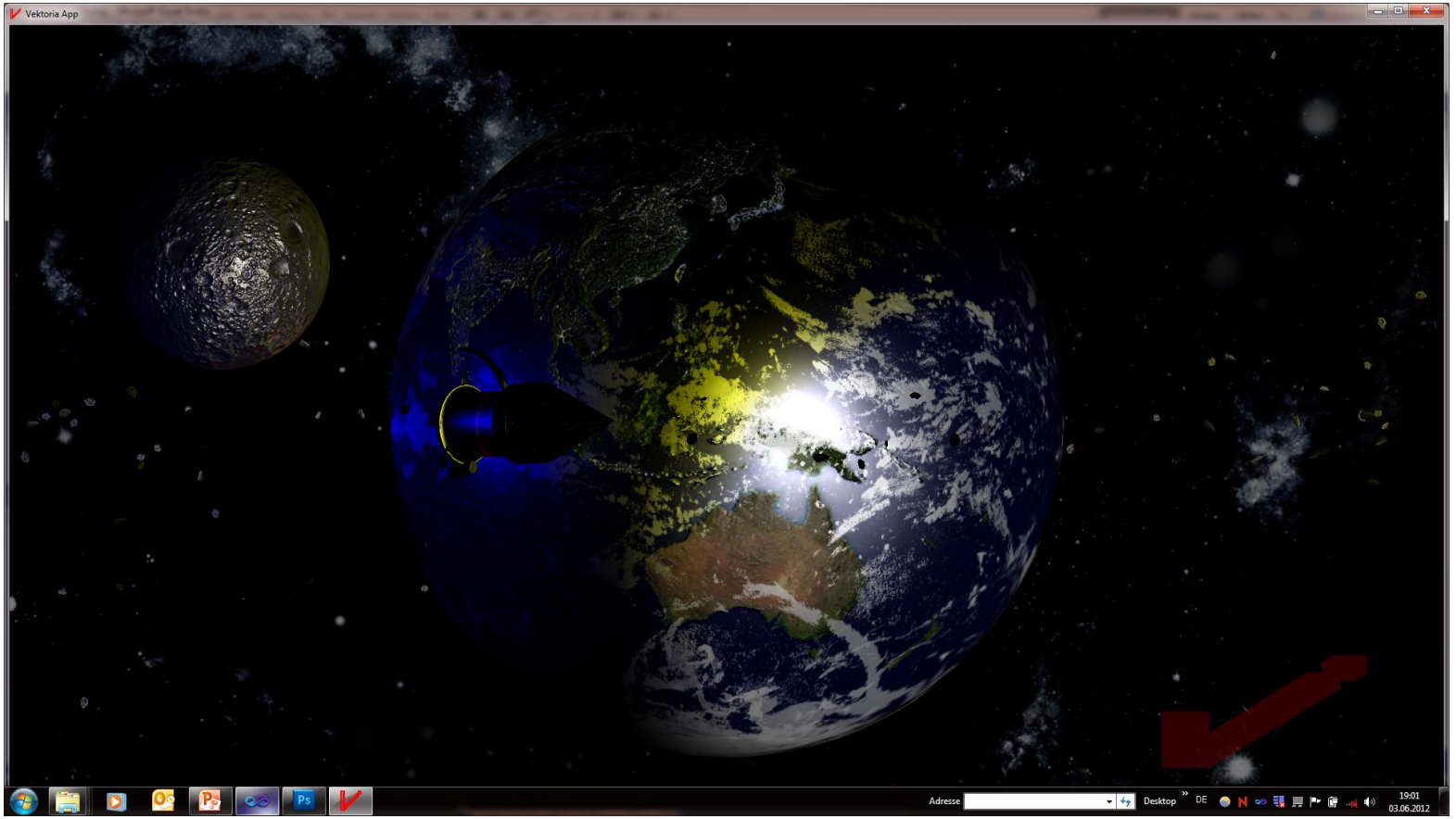
StyleMonochrome (1.0,0.3,0.1)



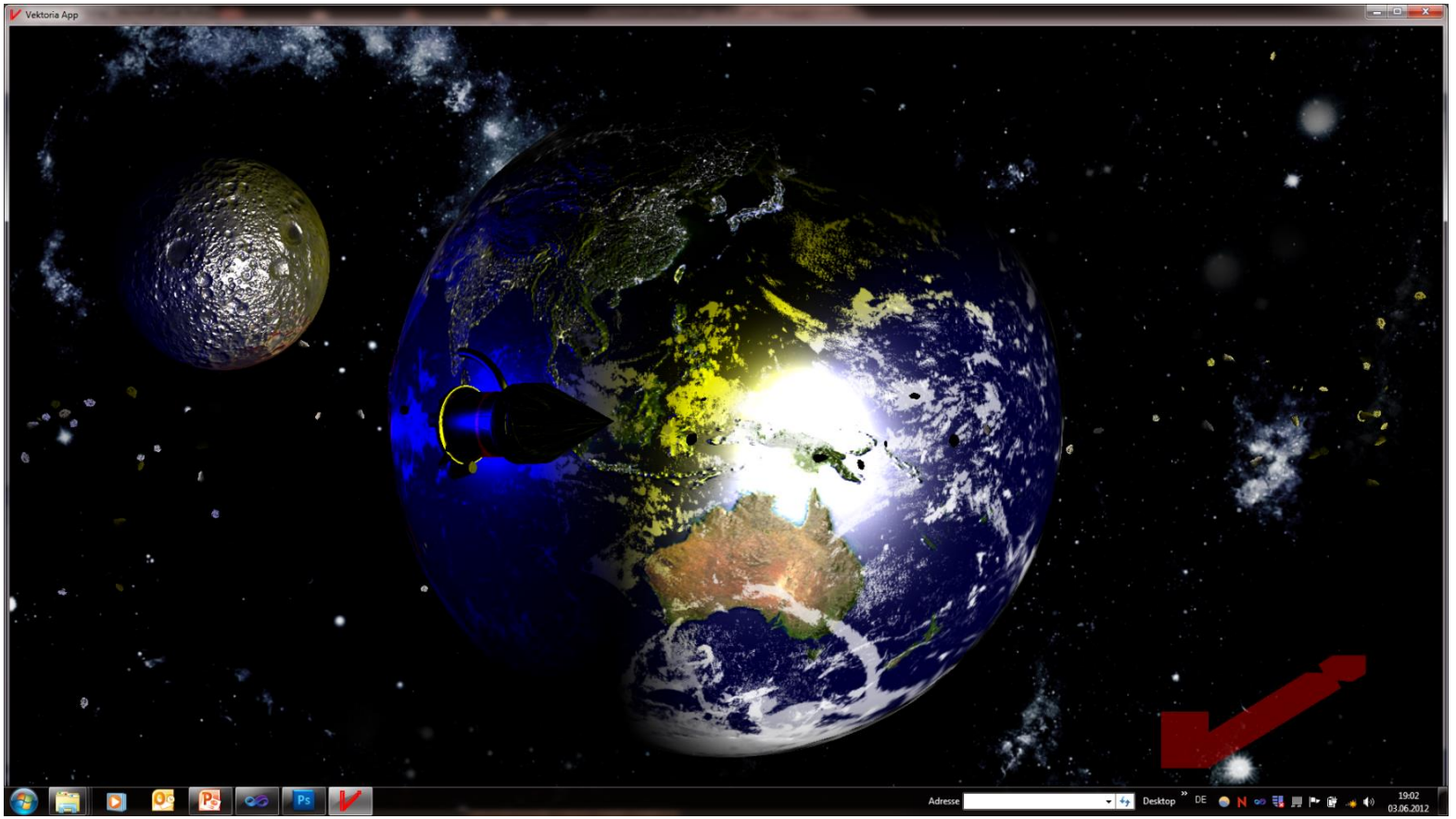
StyleOutlining



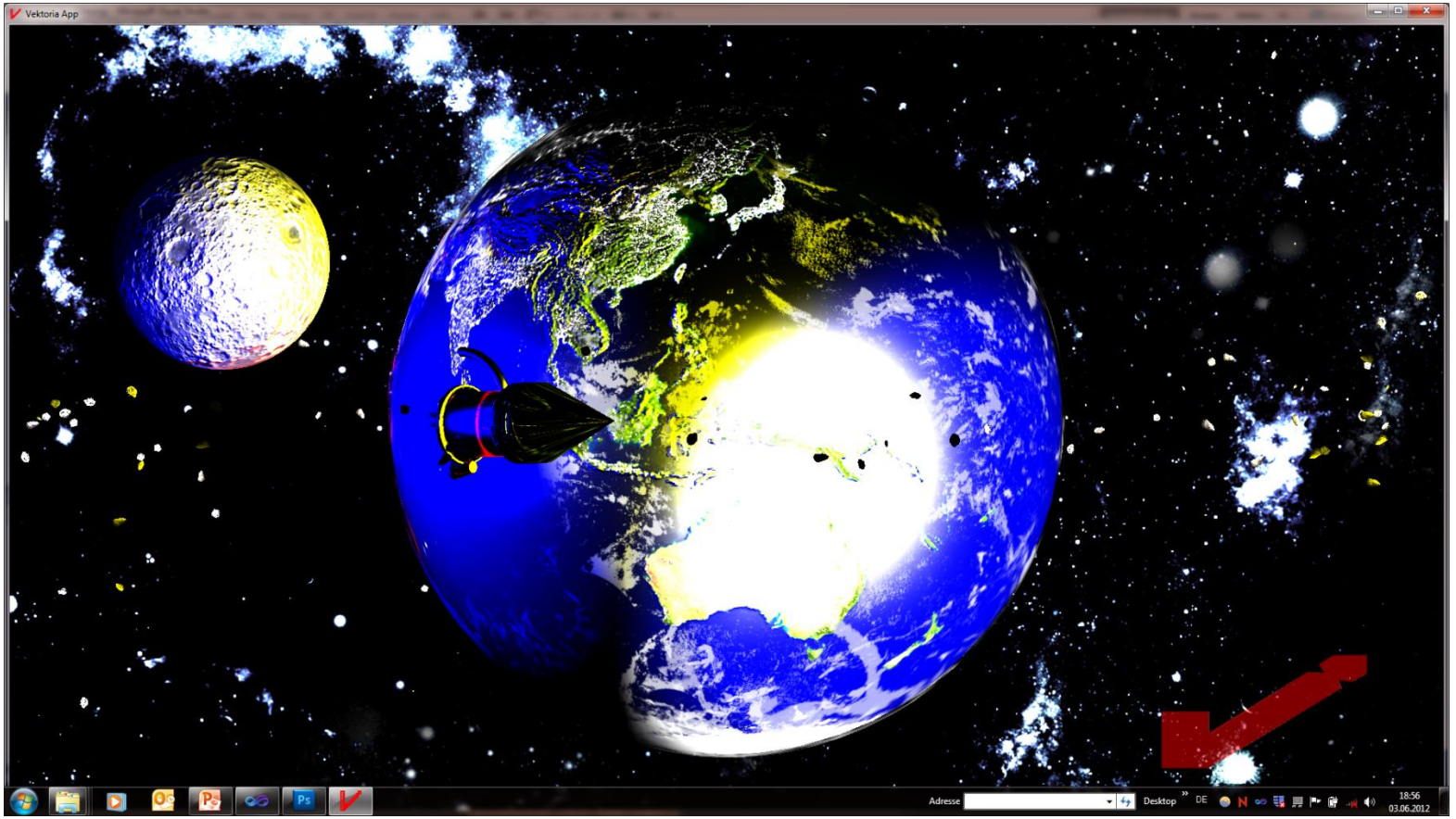
StyleBrightness (0.5)



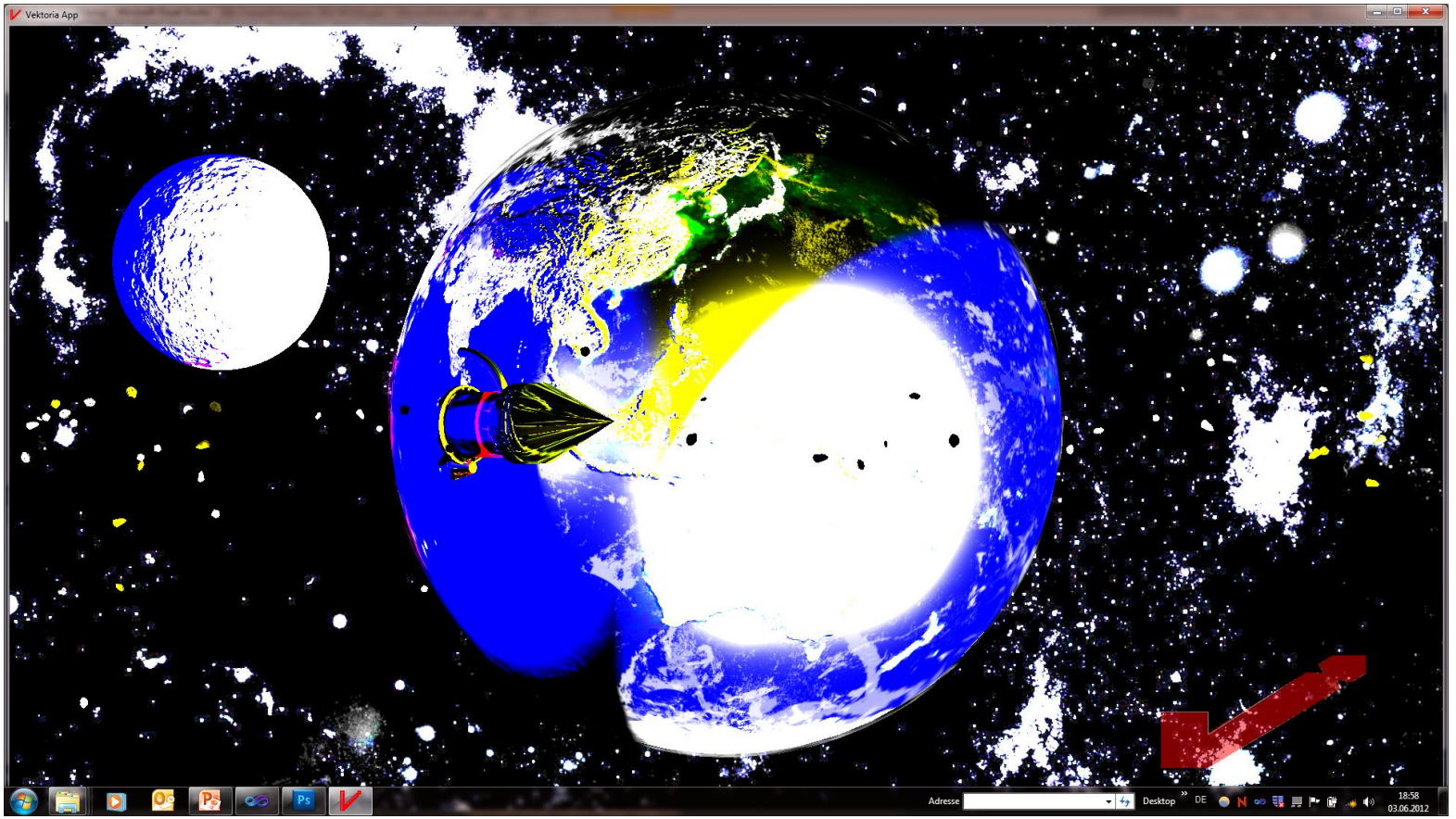
StyleBrightness (1.0)



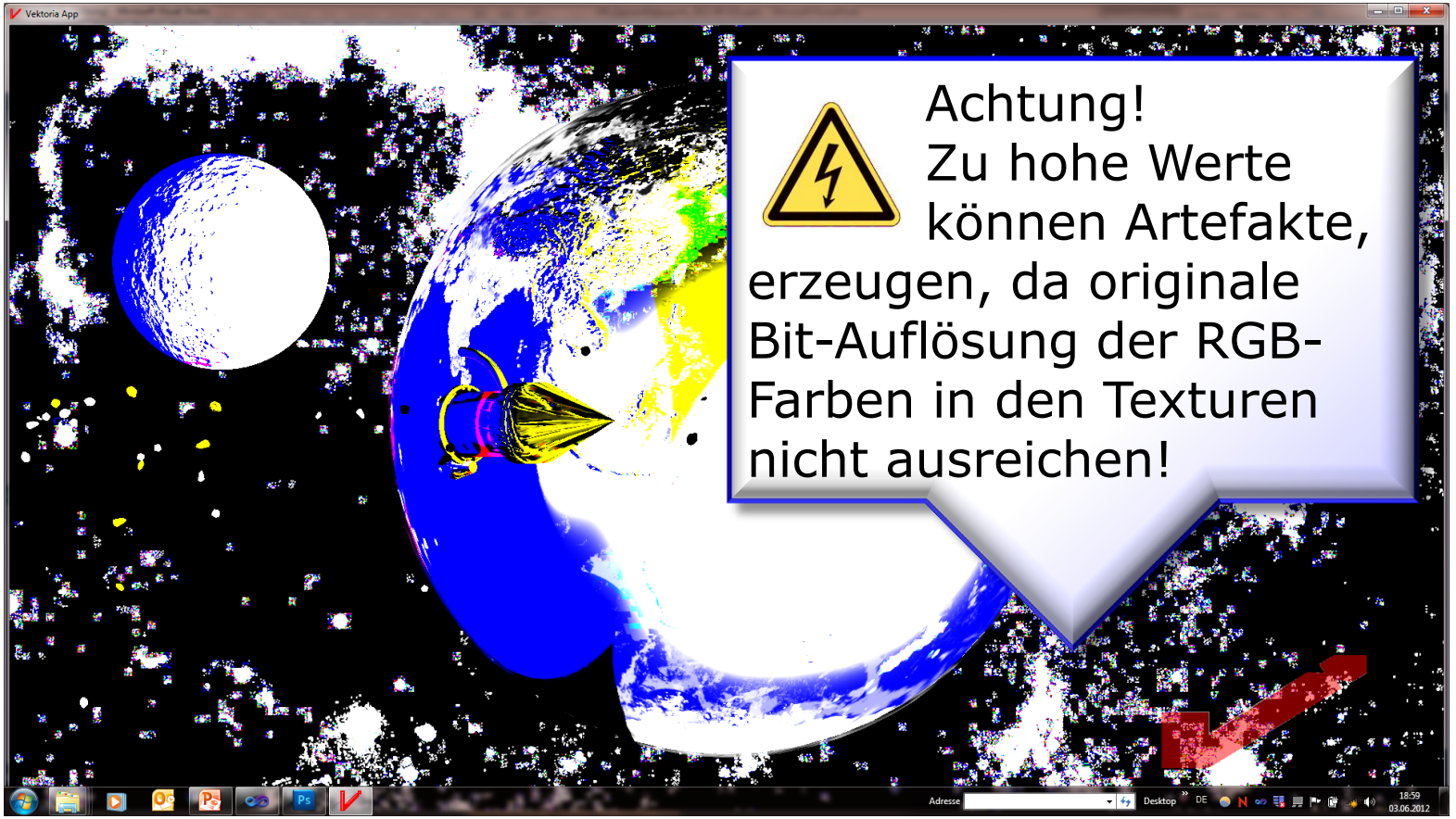
StyleBrightness (5.0)



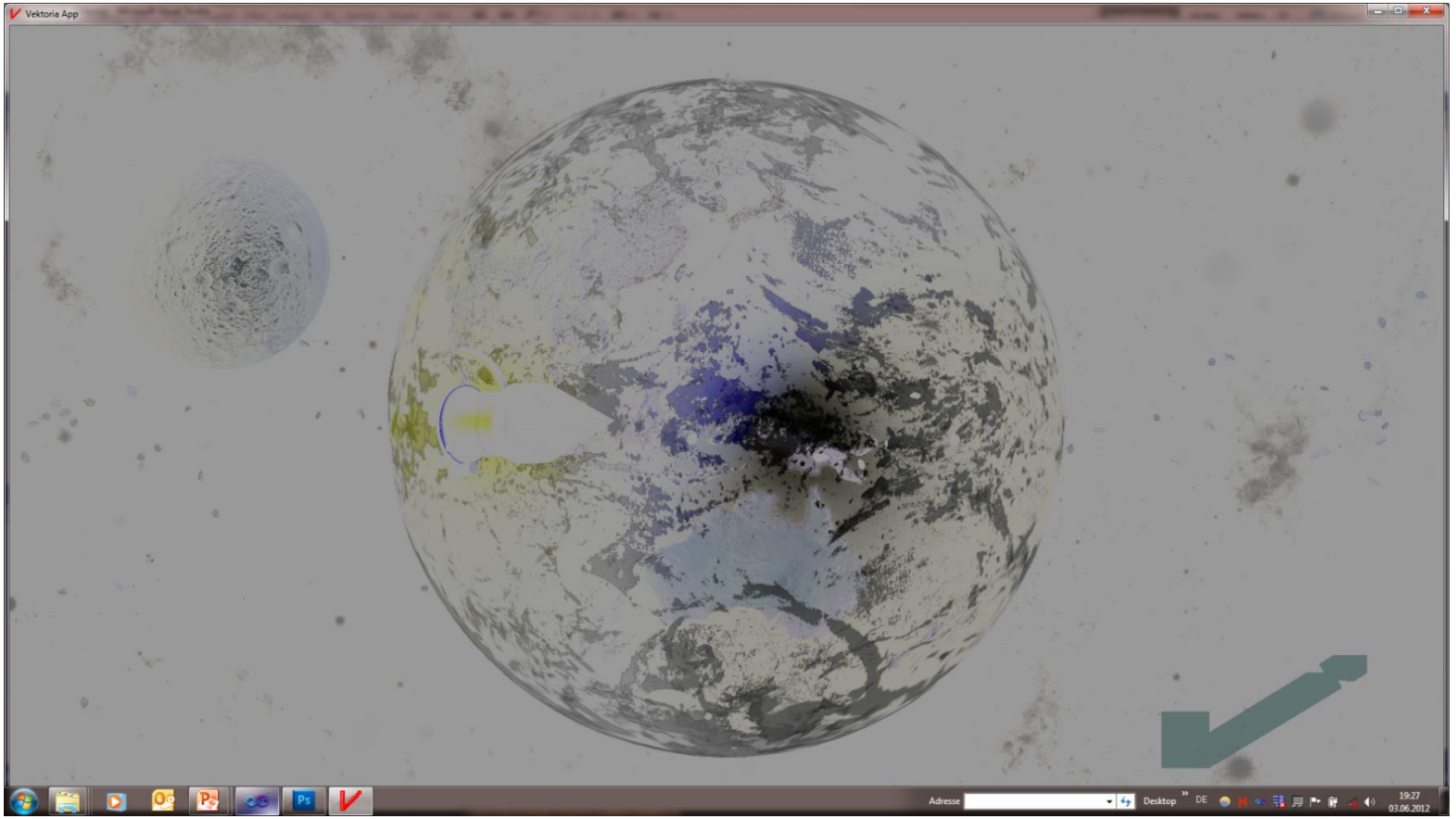
StyleBrightness (50.0)



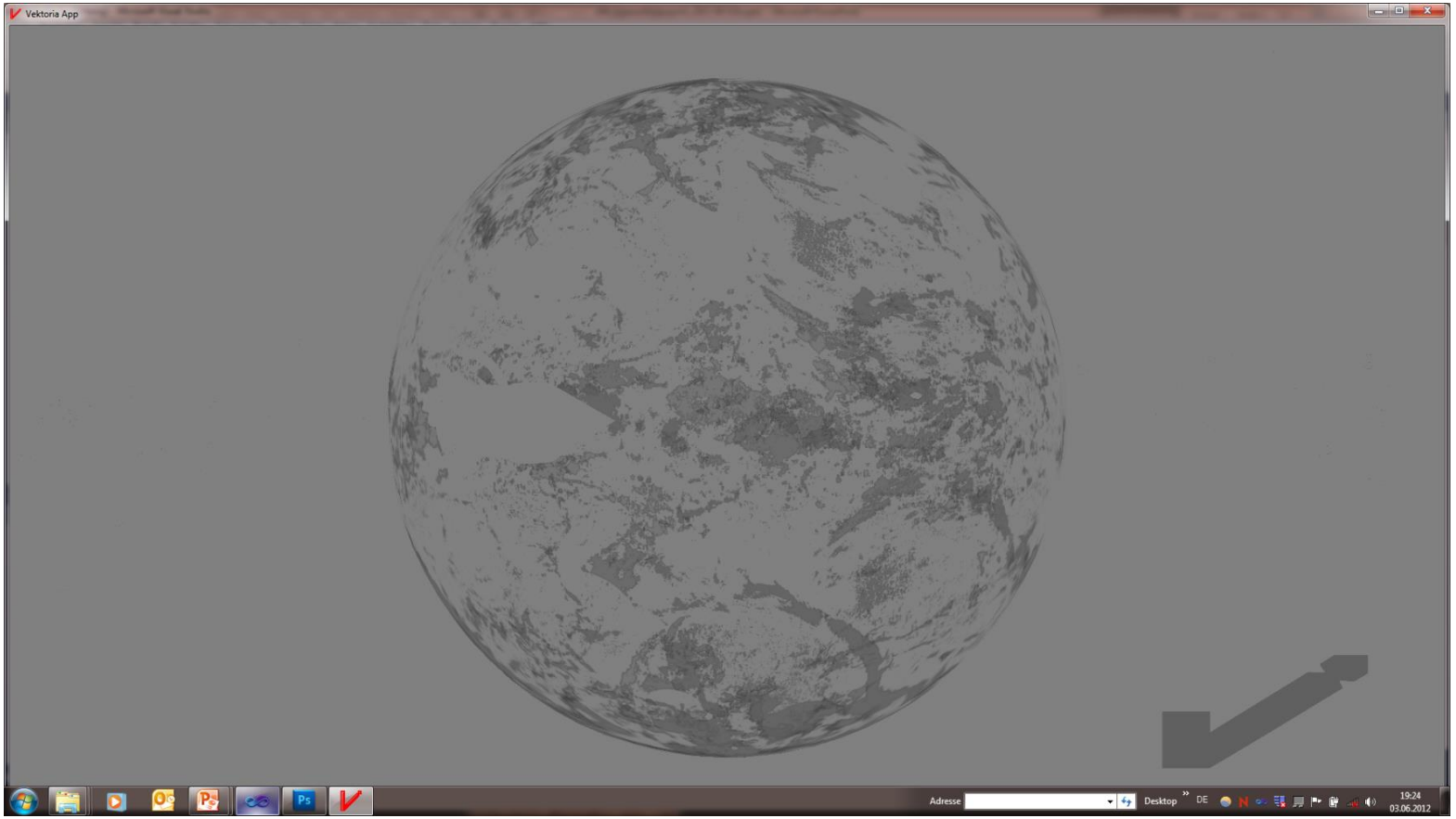
StyleBrightness (500)



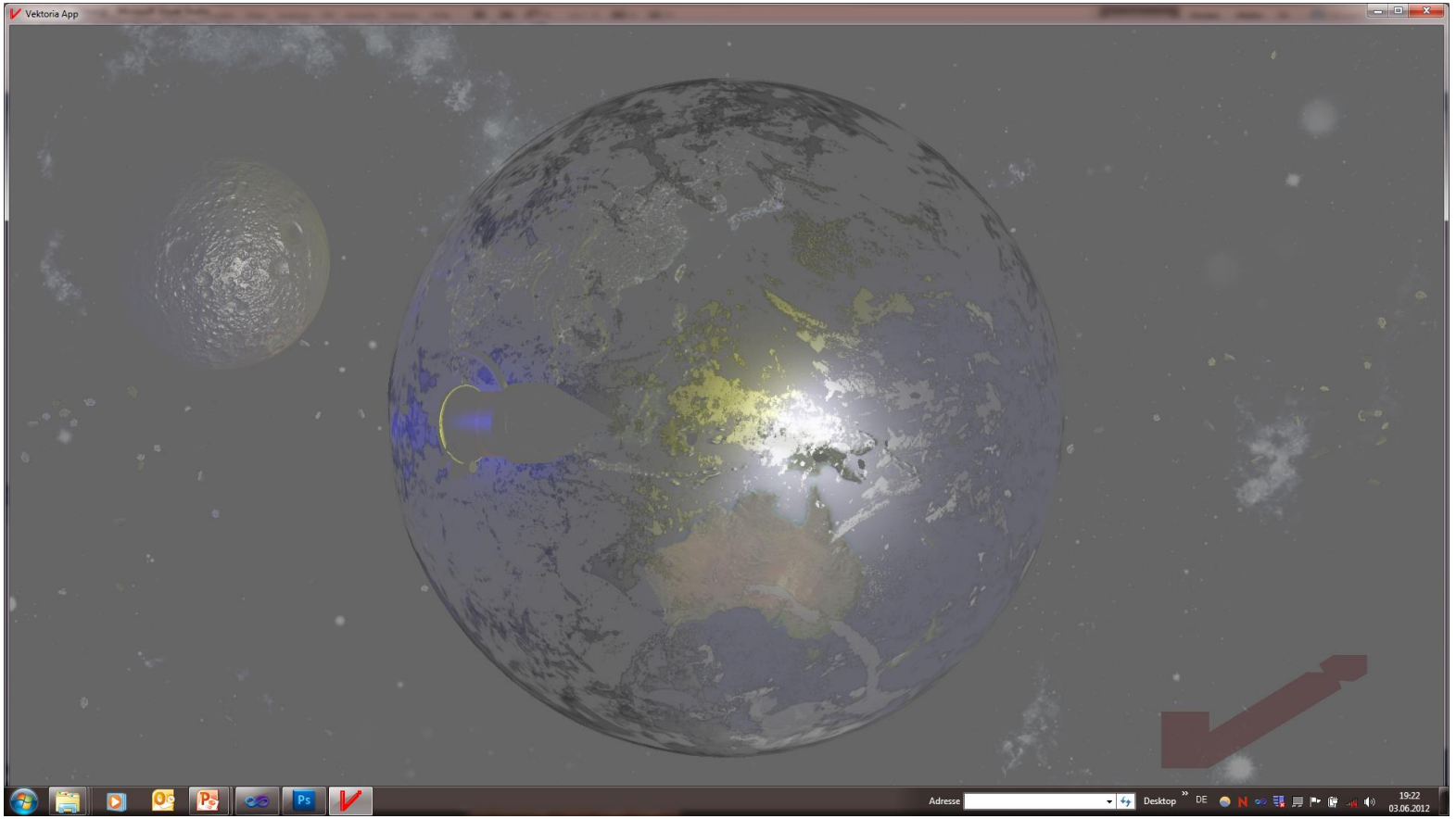
StyleContrast (-0.6)



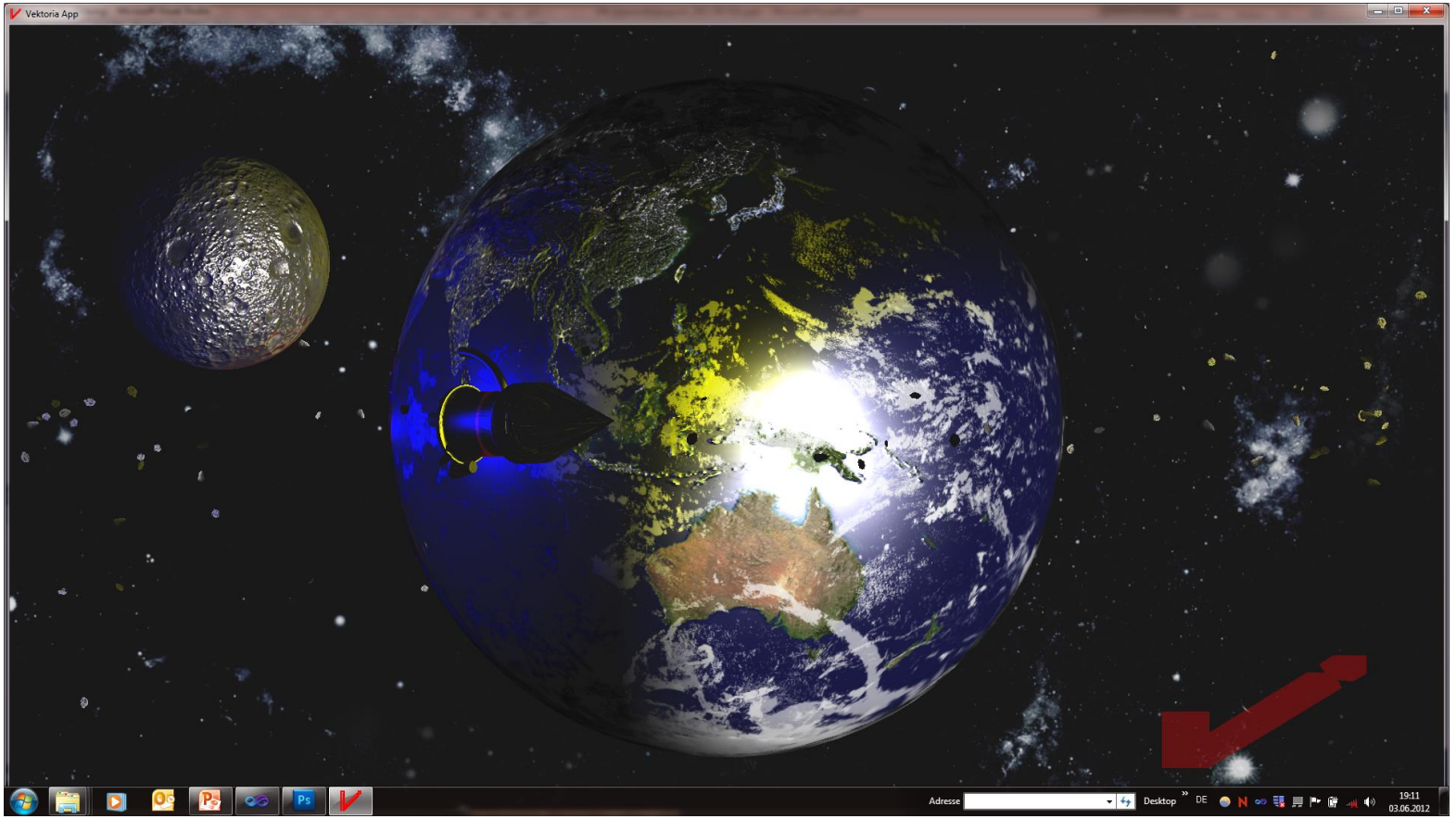
StyleContrast (-0.5)



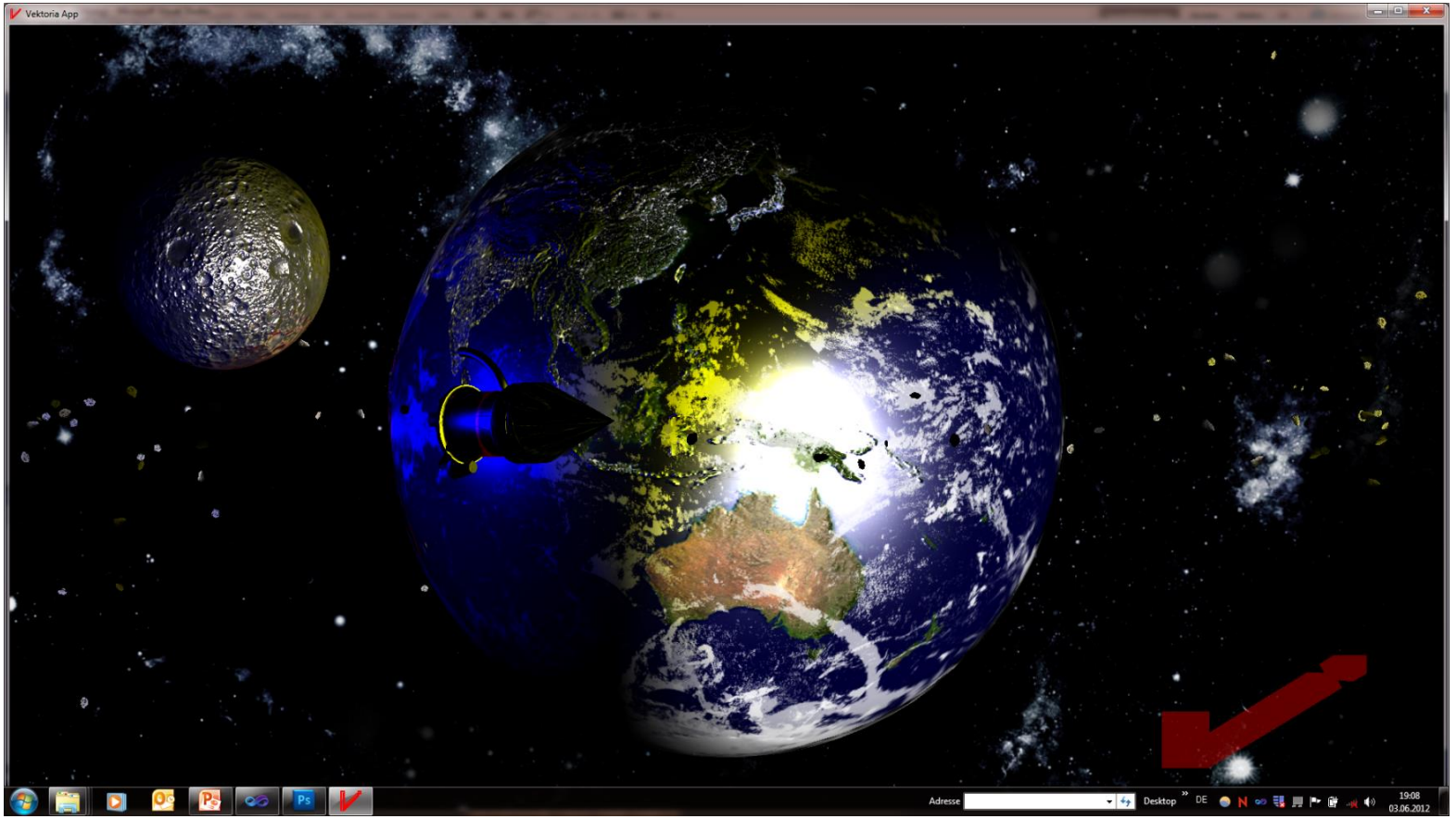
StyleContrast (-0.1)



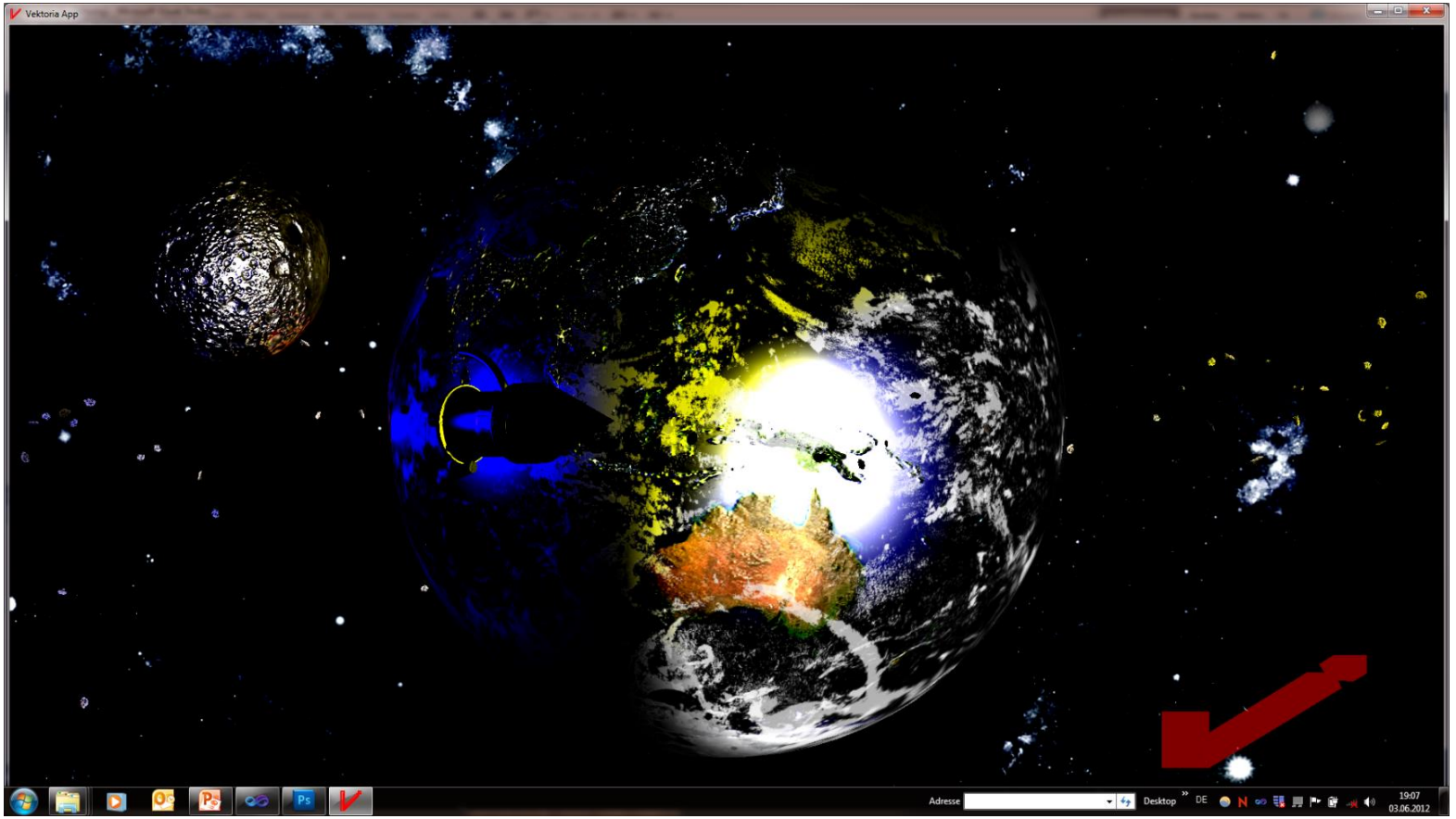
StyleContrast (-0.1)



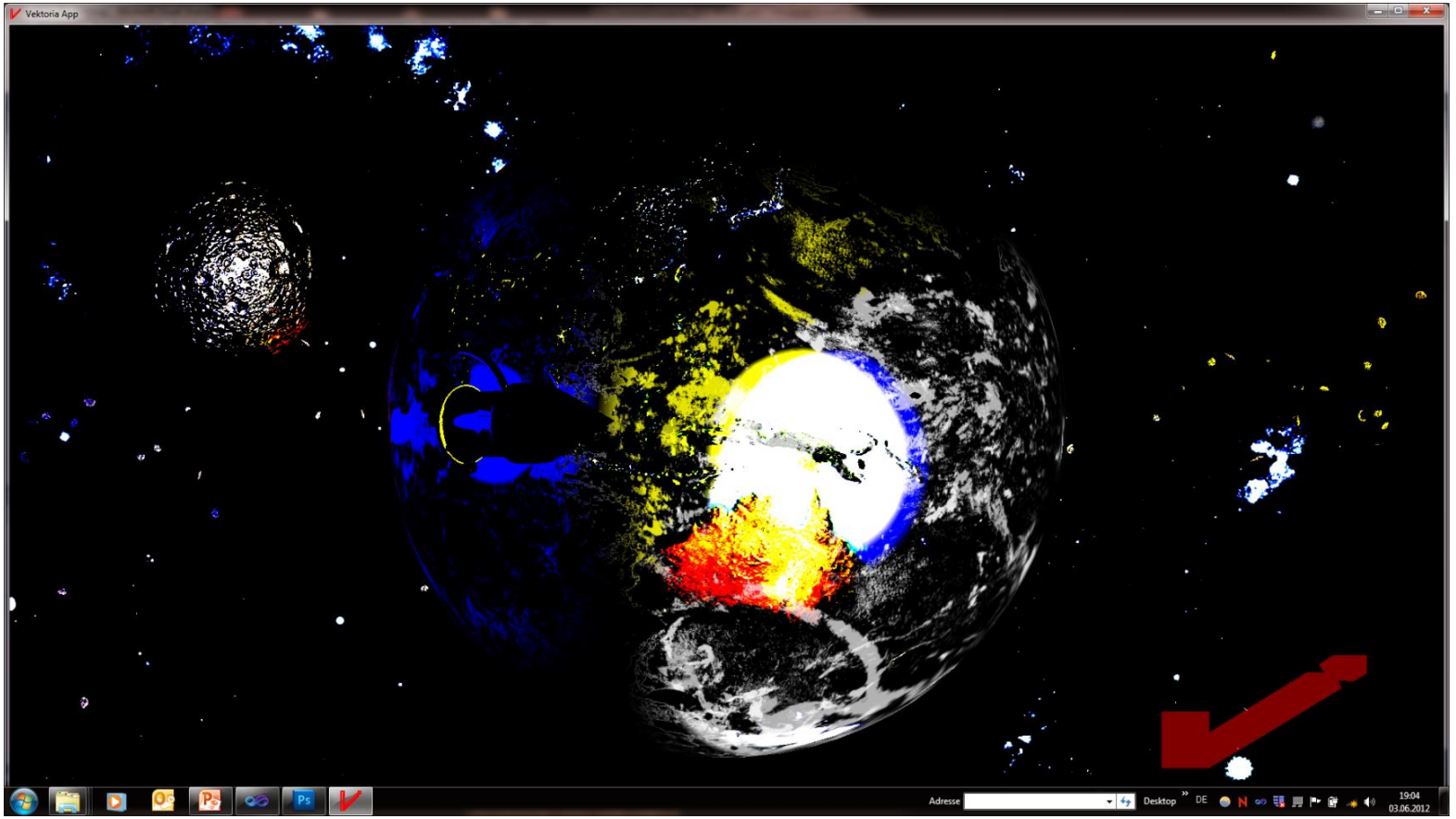
StyleContrast (0.0)



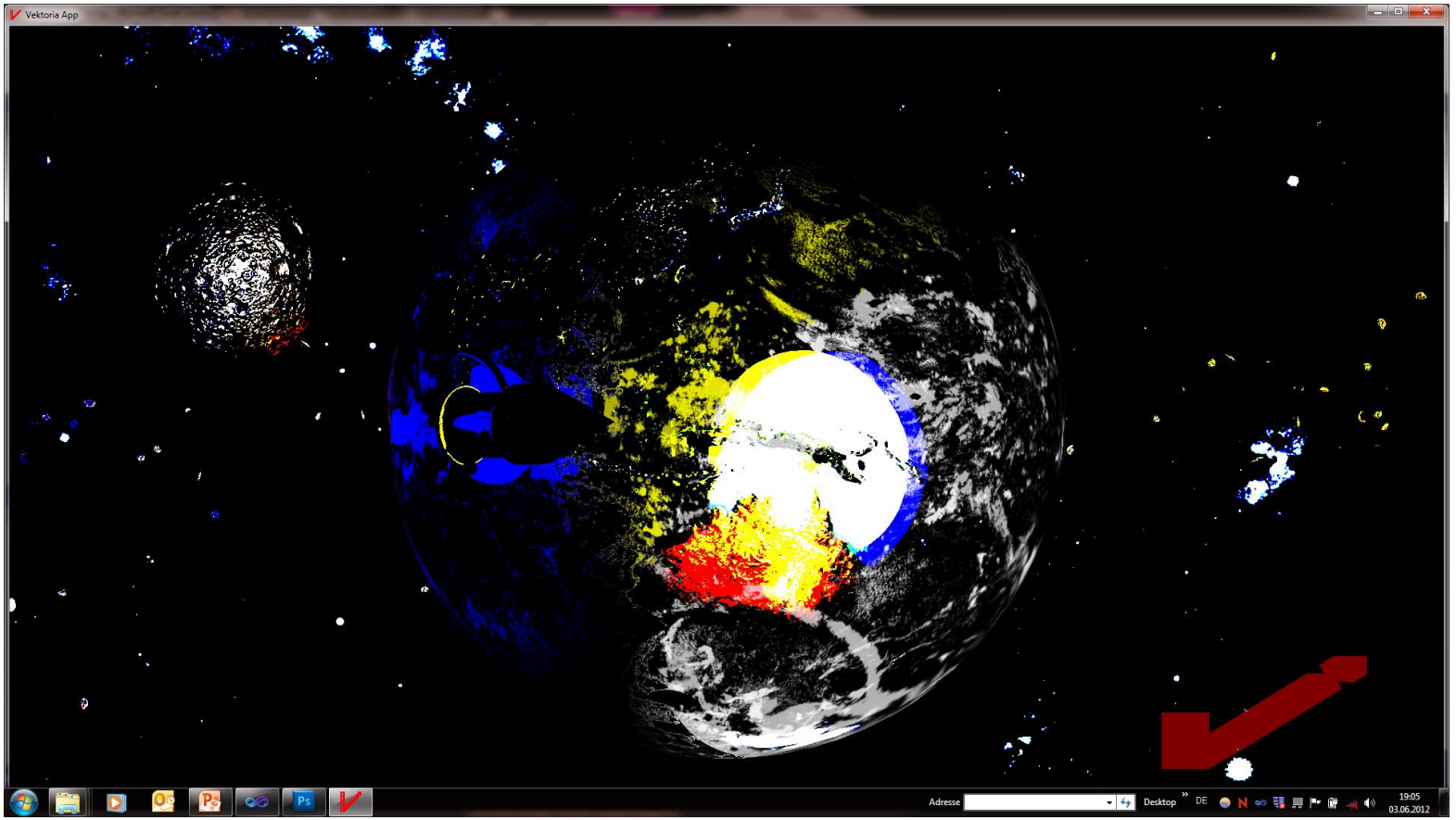
StyleContrast (0.5)



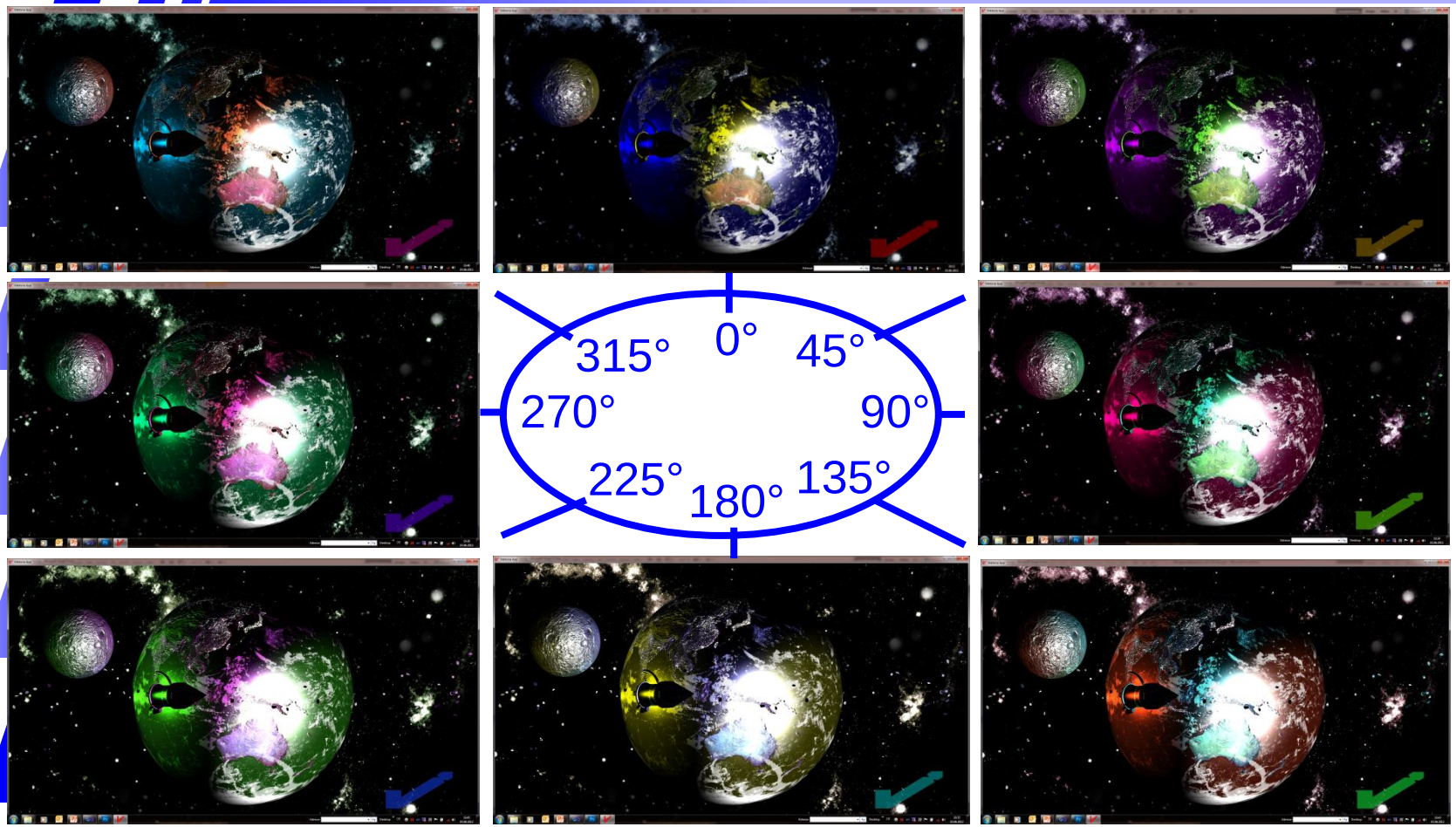
StyleContrast (5.0)



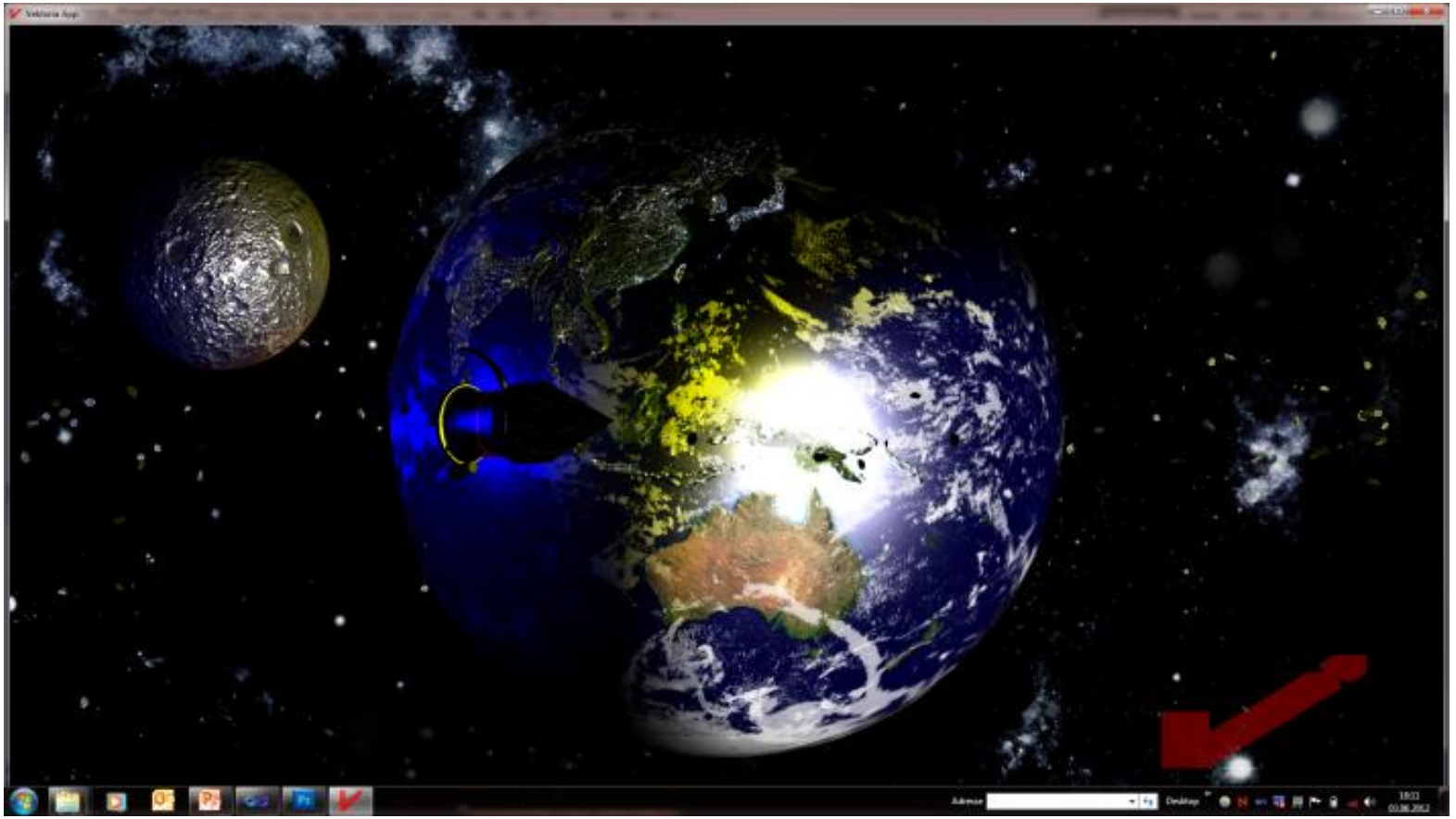
StyleContrast (500.0)



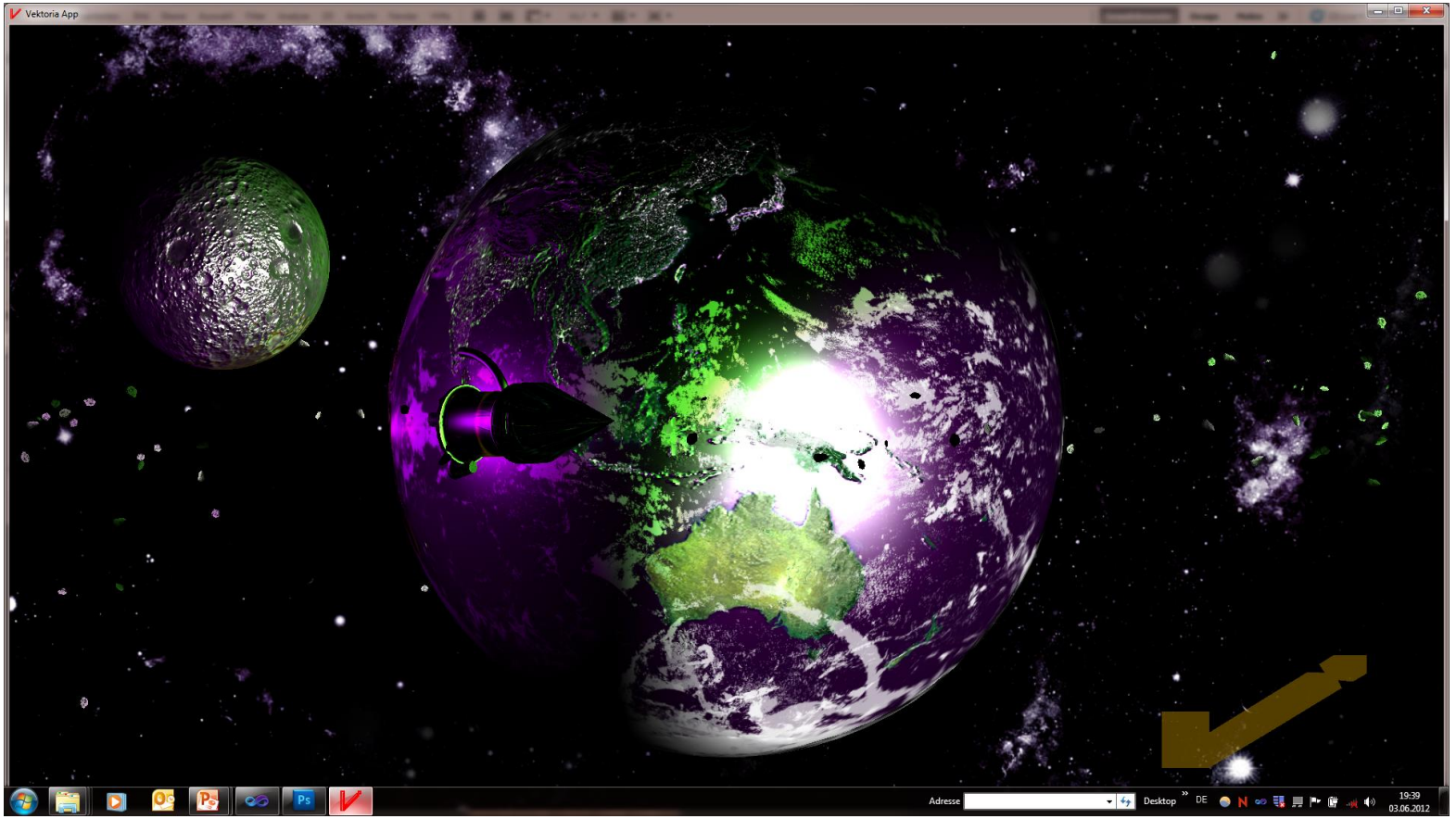
StyleRotateHue



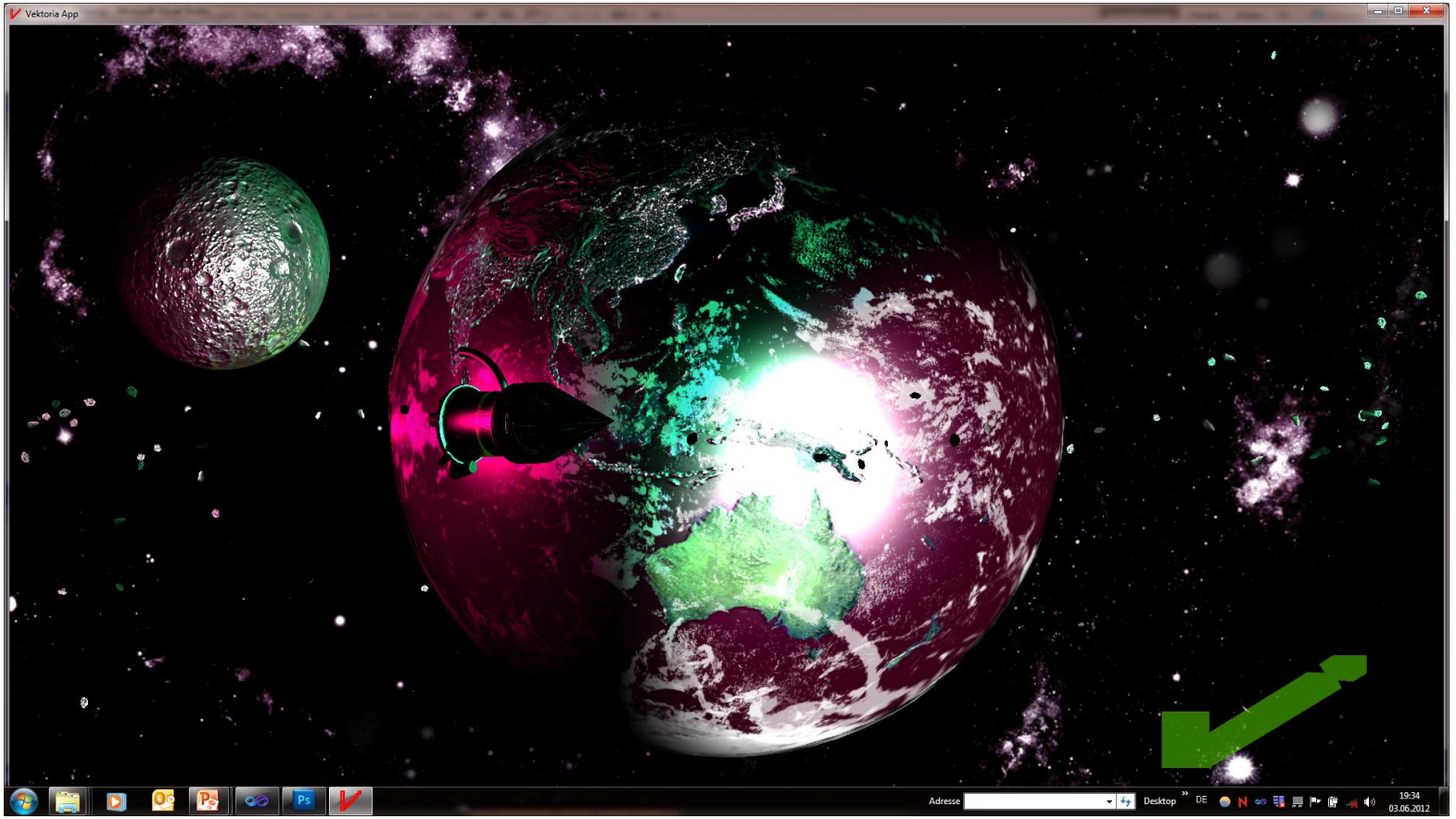
StyleRotateHue(0)



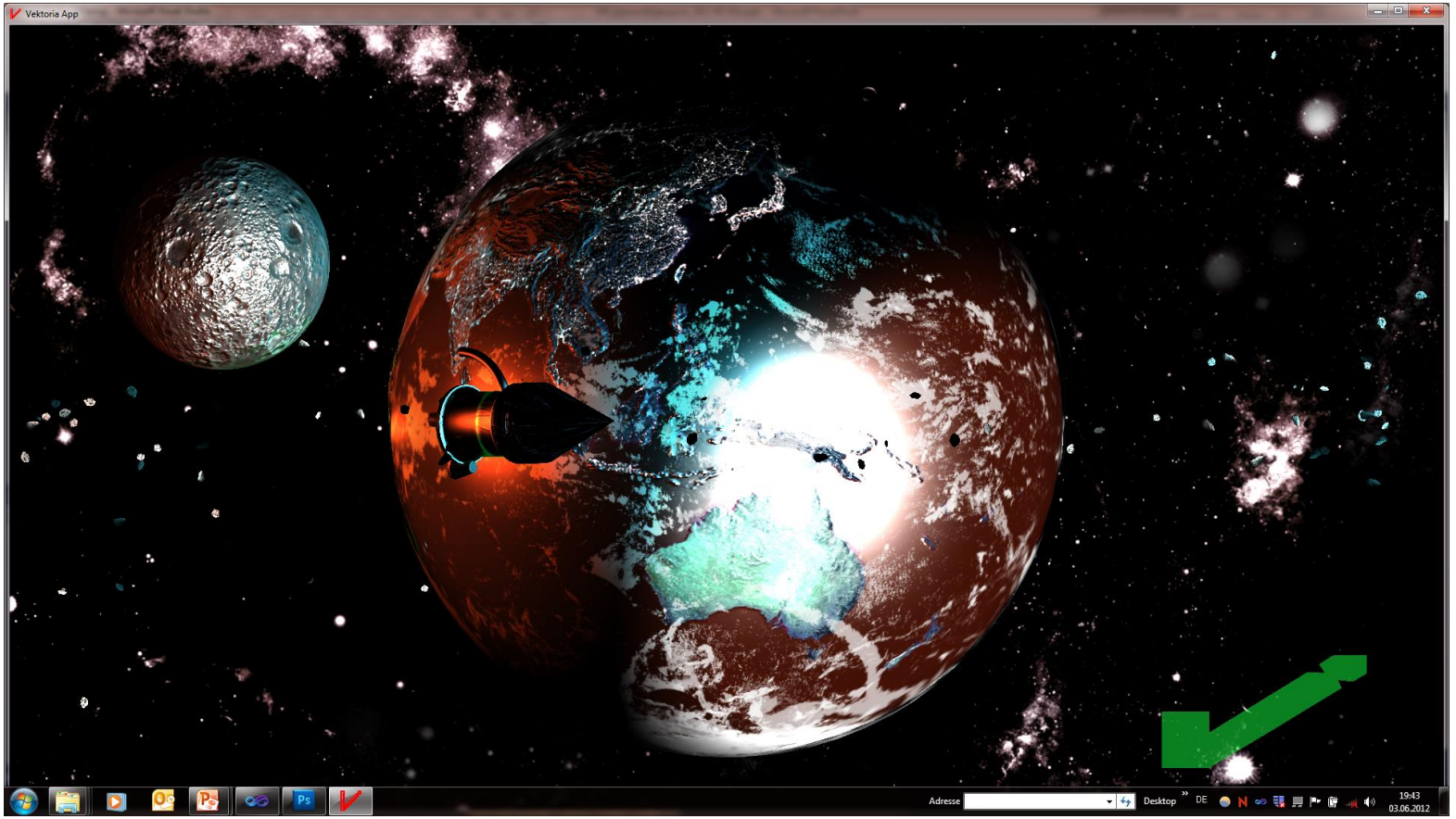
StyleRotateHue(QUARTERPI)



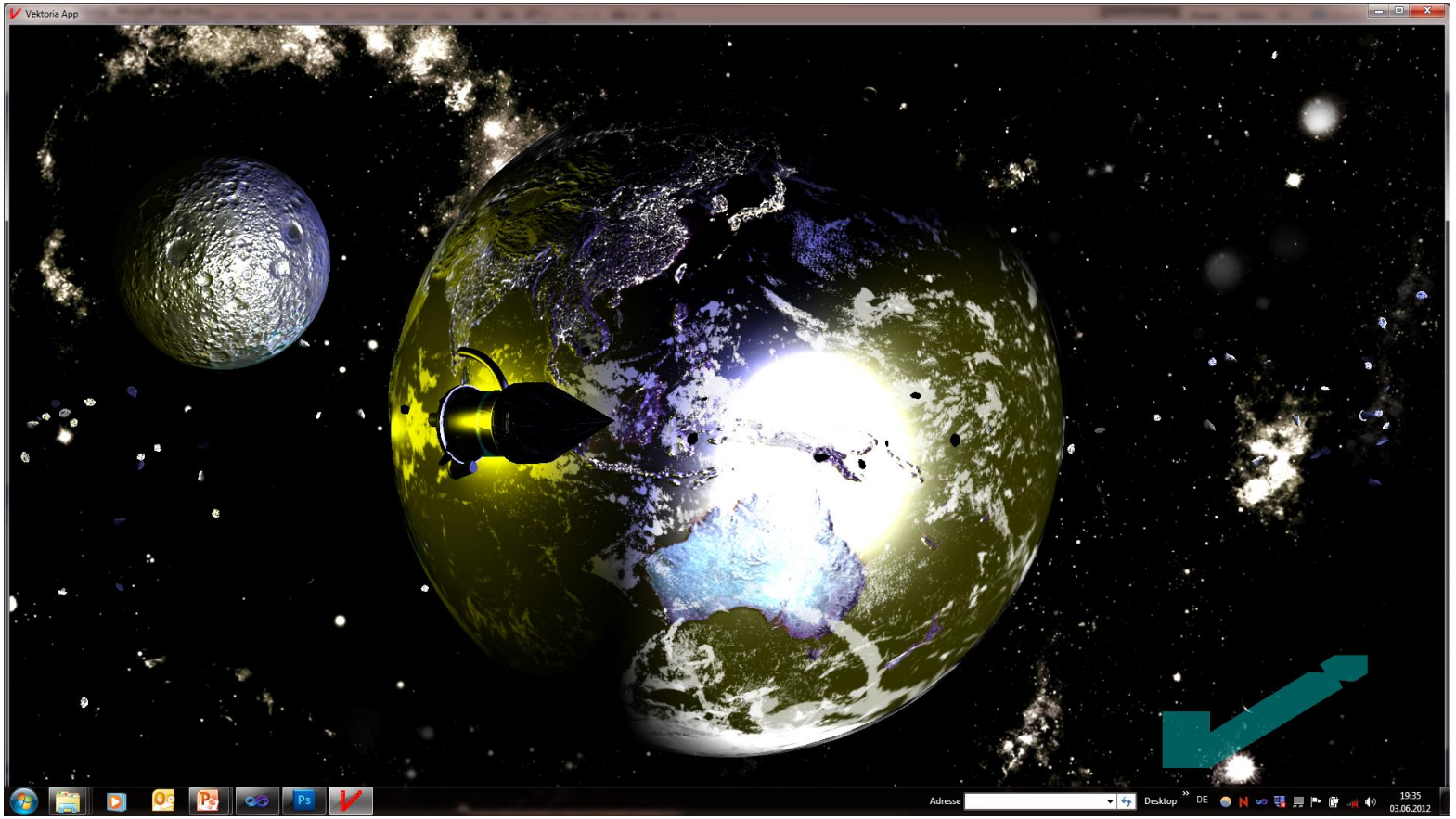
StyleRotateHue(HALFPI)



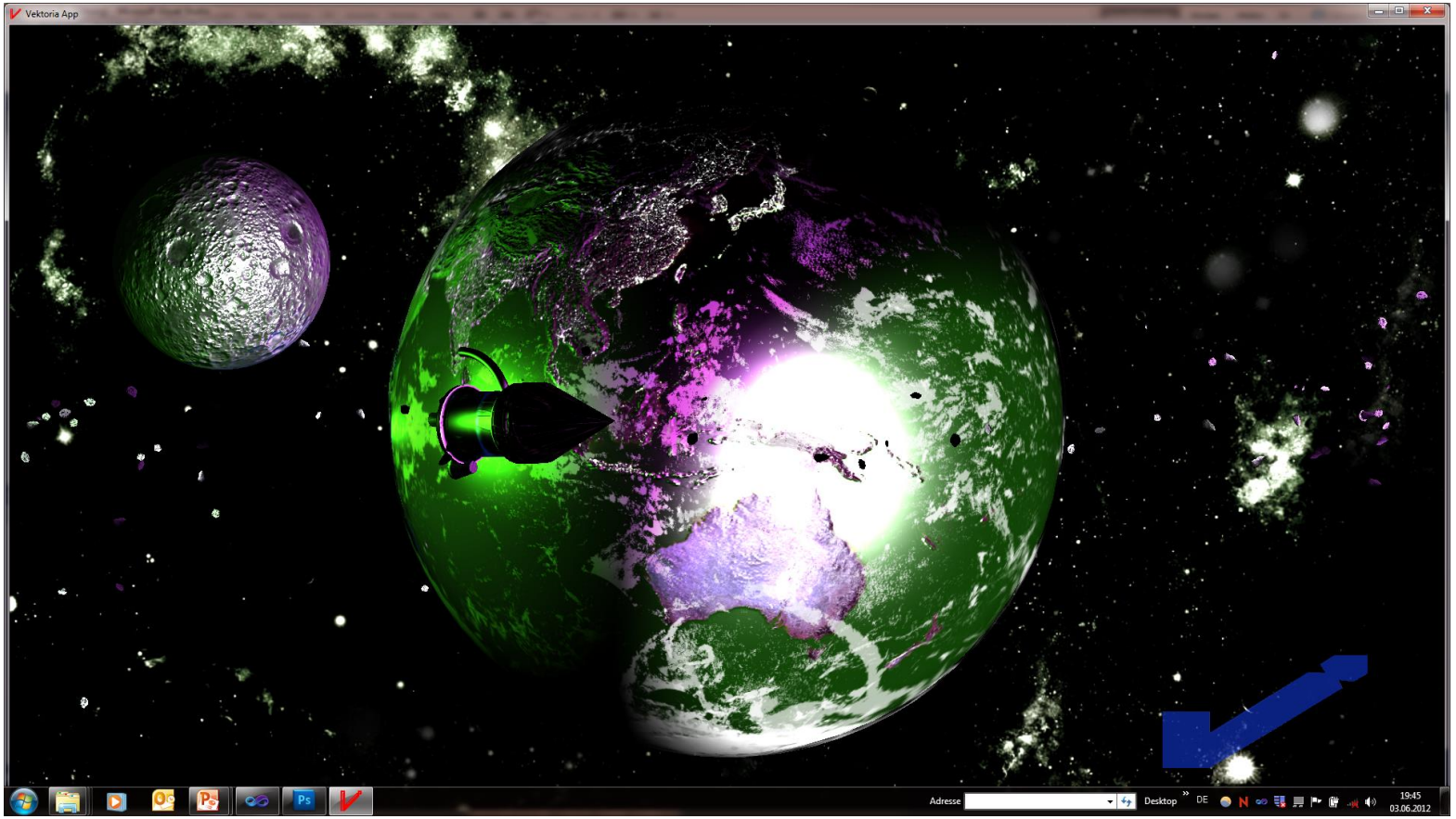
StyleRotateHue(PI*0.75)



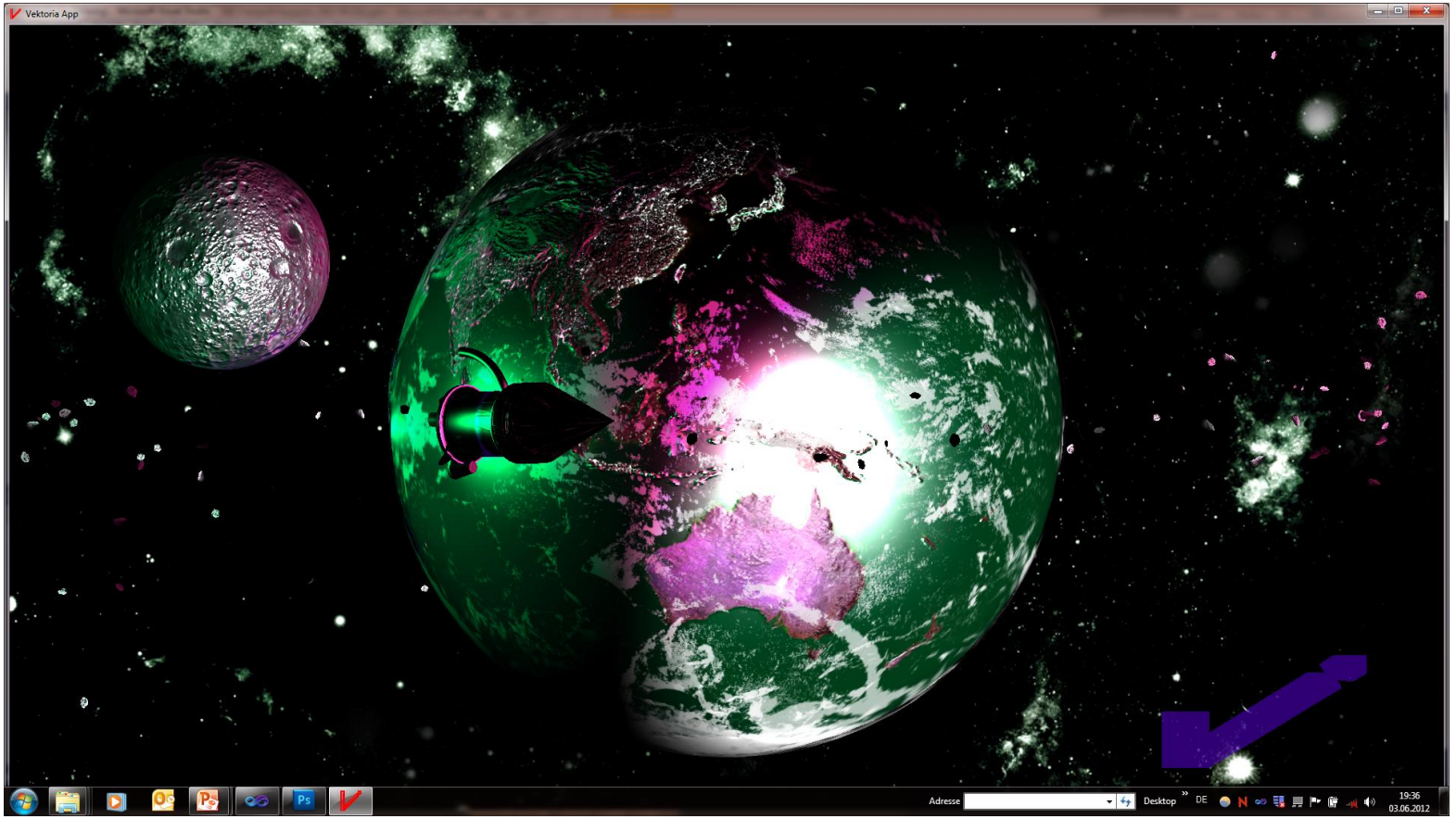
StyleRotateHue(PI)



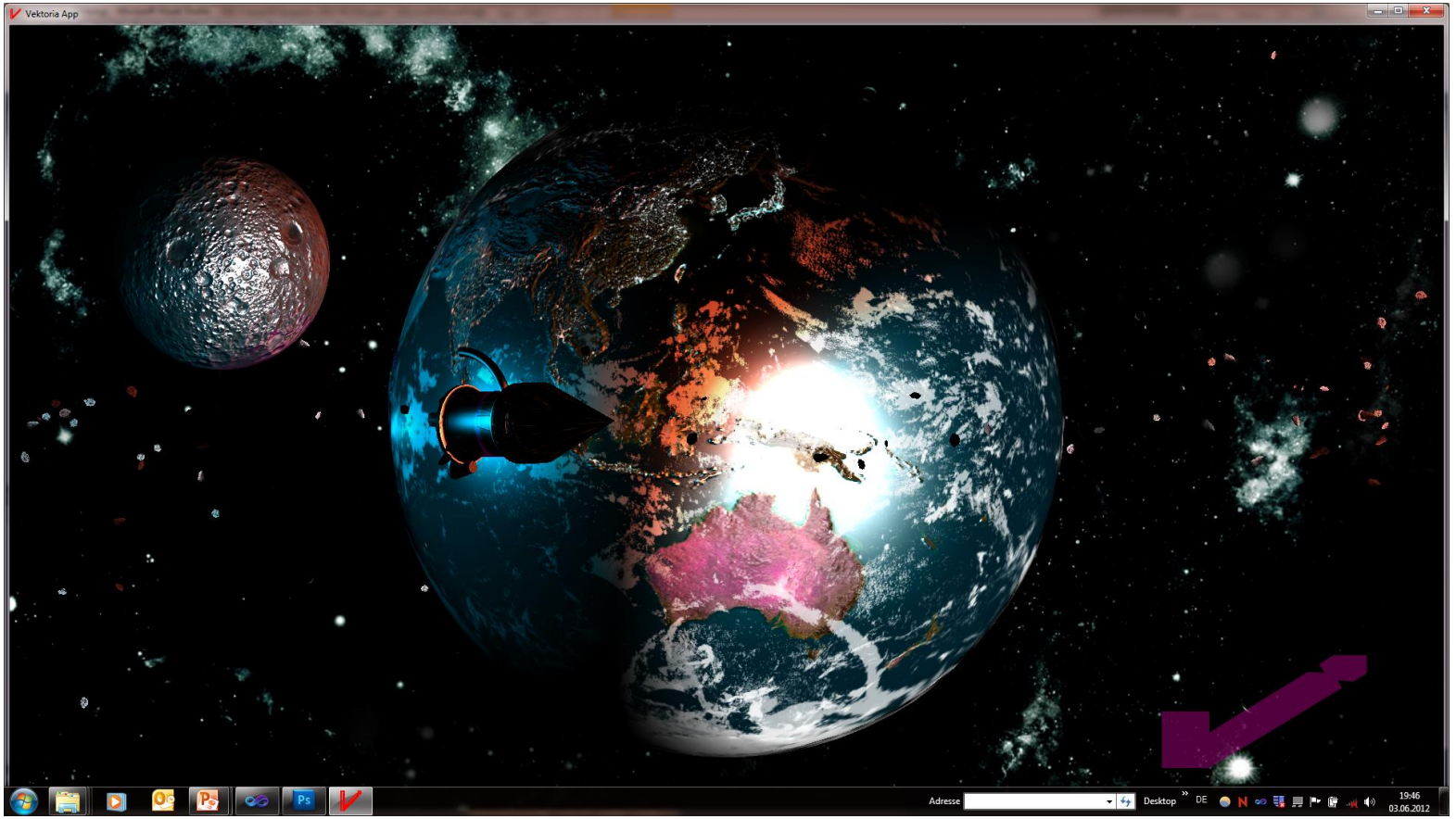
StyleRotateHue(PI*1.25)



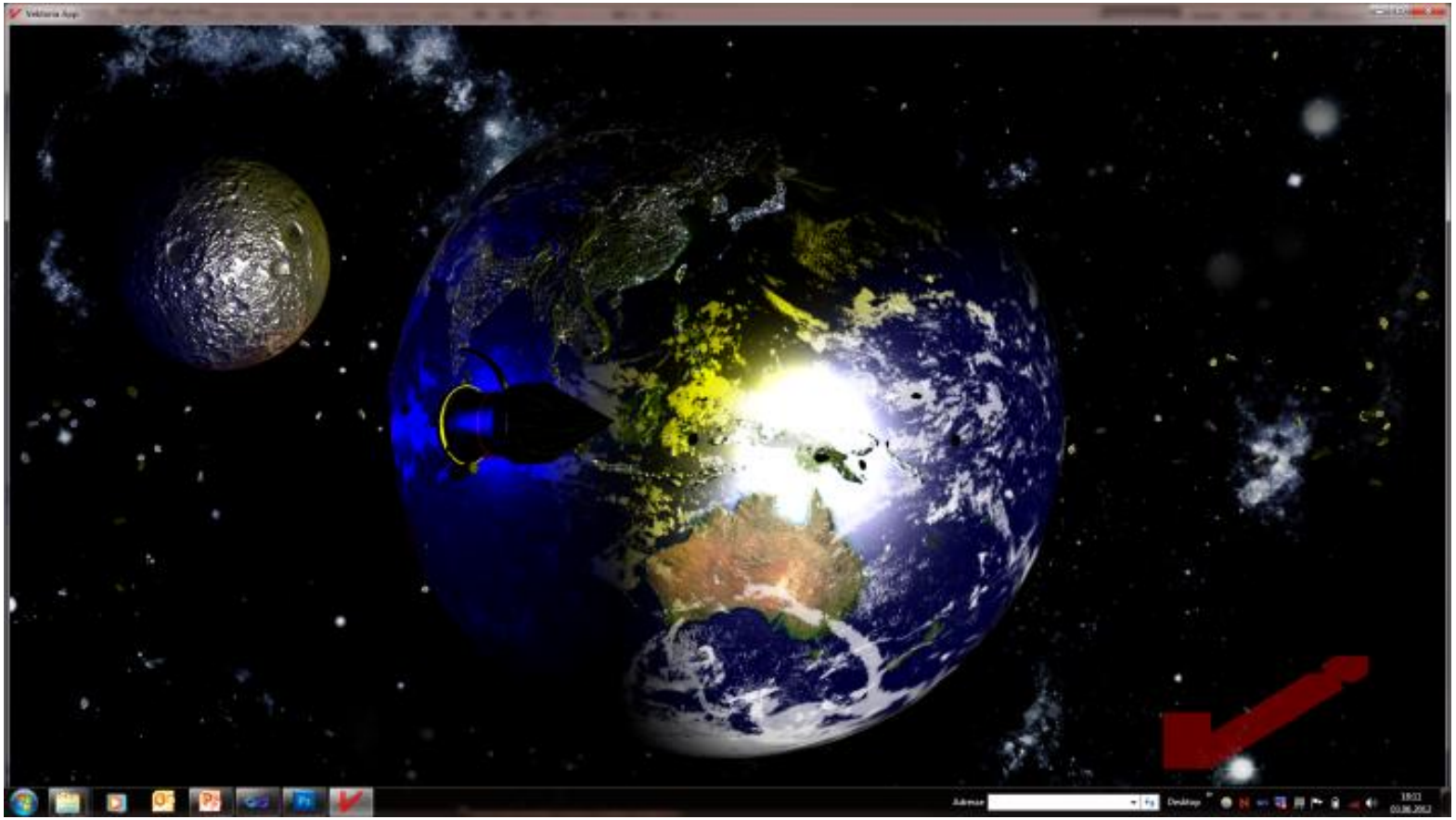
StyleRotateHue(PI+HALFPI)



StyleRotateHue(PI*1.75)



StyleRotateHue(TWOPI)



Übung zu Viewportstilen

Probieren Sie verschiedene Stile aus!

1

2

3

4

5

Freiwillige Zusatzaufgabe für die schnellen Nerds:

- Rotieren Sie während der Laufzeit die Farbgebung der Szene!

StyleOwn()

1. Mit der Methode StyleOwn der Klasse CViewport lässt sich ein eigener Shader-Teil in HLSL programmieren.
2. Der Punkt, in dem man seinen eigenen Code einpflegen kann, ist in der Datei „shader.hlsl“ markiert.
3. Achtung, um StyleOwn zu benutzen muss im Compiler #
4.
5.

Shadersprachen

1. GLSL (Aussprache: „tschielsläng“)

   OpenGL Shading Language

   OpenGL-spezifischer C-Dialekt

2. Cg

   C for Graphics

   Von NVidia entwickelter C-Dialekt, der auch von anderen Grafikkartenherstellern unterstützt wird. Ist sowohl für OpenGL als auch Direct3D einsetzbar

3. HLSL

   High Level Shading Language:

   Microsofts Implementierung von Cg, ist in Direct3D enthalten

Gg Profile

- GPU-Programmierung ist noch nicht auf der selben Abstraktionsstufe wie die CPU-Programmierung angelangt.
- Jeder Grafikprozessor hat stark unterschieliche Fähigkeiten
- Die Basis-APIs haben unterschiediche Schnittstellenmöglichkeiten

⇒ Lösung: Profilangabe

⇒ Ein Cg-Profil ist eine Untermenge von Gg

Gg Profile

Basis-API	VS-Profil	FS-Profil
OpenGL ARB	arbvp1	arbfp1
OpenGL NV40	vp40	fp40
OpenGL NV30	vp30	fp30
OpenGL NV2X	vp20	fp20
DirectX 11.0	vs_5_x	ps_5_x
DirectX 10.0	vs_4_x	ps_4_x
DirectX 9.0	vs_2_x	ps_2_x
DirectX 8.0	vs_1_1	ps_1_1, ps_1_2, ps_1_3

Cg/HLSL elementare Datentypen

bool	Wahrheitswert	
int	32bit Ganzzahl	
uint	32bit unsigned int	(nur bei HLSL)
fixed	12bit Fixkommazahl	(1-1-10)
half	16bit Gleitkommazahl	
float	32bit Gleitkommazahl	
double	64bit Gleitkommazahl	
float2	2D-Gleitkommavektor	
float3	3D-Gleitkommavektor	
float4	4D-Gleitkommavektor	
matrix	Homogene 4x4-Matrix	(nur bei HLSL)
float4x4	Homogene 4x4-Matrix	(nur bei Cg)

Cg/HLSL komplexe Datentypen

sampler	2D-Texturobjekt (nur bei Cg)
Texture1D	1D-Texturobjekt (nur bei HLSL)
Texture2D	2D-Texturobjekt (nur bei HLSL)
Texture3D	3D-Texturobjekt (nur bei HLSL)
struct	Union
[]	Array
[][]…	multidim. Array
vector<datent, i>	Vektor aus i Variablen des Datentyps d (nur bei HLSL)

Beispiele:

vector<uint, 4>	Vektor aus 4 unsigned integers (HLSL)
vector<float, 8>	Vektor aus 8 floats (HLSL)

Cg-Parameterarten (Beispiele)

1

Eingabeparameter

- `in float3 i_f3Pos : POSITION;`
- `in float4 i_f4Color : COLOR;`

2

Ausgabeparemeter

3

- `out float3 o_f3Pos : POSITION;`
- `out float4 o_f4Color : COLOR;`

4

Konstante EIngabeparameter

- `uniform float fTime;`
- `uniform float4x4 f44Rotation;`

5

Mathematische Cg/HLSL-Funktionen

- mul (A,B) — Multiplikation von Vektoren und Matrizen (überladene Funktion)
- abs(A) — Absolutfunktion
- log2(A) — Zweierlogarithmus
- sqrt(A) — Wurzel
- pow(A,B) — Potenzfunktion
- saturate(A) — Schneidet Wert ab, so dass er zwischen 0 und 1 liegt (nur in HLSL)
- …

Kein #include notwendig!

Cg/HLSL-Texturfunktionen

- Sample (*texture2D*, *float2*) (Nur in HLSL)
- texCUBE (Nur in Cg)
- tex3Dproj (3D-Textur-> 2D)
- *texture2D*.GetDimensions(*Bittiefe*,*Breite*,*Höhe*, *Ebenen*)
- …

Kein #include notwendig!

Cg/HLSL-Vektorfunktionen

- dot(V,V) Dot-Produkt
- normalize(V) Vektornormalisierung
- mul(V,M) Vektor-Matrix-Multiplikation
- …

Ansonsten normale Operatoren verwenden (*, +, /, etc.)

Kein #include notwendig!

Was Cg/HLSL mit C gemeinsam hat

structs	**struct**
Arrays, mehrdimensionale Arrays	**[]**, **[2][3]**, …
arithmetische Operatoren	**+, *, /, %, …**
Boolsche Operatoren	**||, &&, !, …**
Increment- und Decrementoperatoren	**++, --**
Konditionaloperator	**?:**
Zuweisungsoperatoren	**= , +=, *=, /=, %=,…**
Kommaoperator	**,**
Benutzerdefinierte Funktionen	**void MyFkt(float f){}**
Einfacher Verzweigungskonstruktor	**if**
Schleifenkonstrukte	**do, while,**
Abbruchkonsrtuktoren	**break, continue**
Kommentare	**// … /* … */**
Einige Präprozessorkonstrukte	**#include, #define, #ifdef, #ifndef,…**

Was Cg/HLSL nicht bereitstellt

- Mehrfachverzweigungen **switch**
- Sprünge **goto**
- Pointer *****
- chars und strings **char, char ***
- Keine rekursiven Funktionen möglich!
- Da C und nicht C++, dürfen natürlich keine objektorientierten Konstrukte verwendet werden (**class, public, private :: etc**.)

Vorgefertigte Konstruktoren

Vorgefertigte Konstruktoren, z.B::

- float4 f4 = float4(1.0, 3.0, 2.0, 0.5);
- float3 f3Normale1 = float3(4.0, -2.0, 5.0);
- float2 f2UV = float2(0.5, 1.0);

Swizzling

Vorgefertigte Konstruktoren, z.B::

- float4 f4 = float4(1.0, 3.0, 2.0, 0.5);
- float3 f3Normale1 = float3(4.0, -2.0, 5.0);
- float2 f2UV = float2(0.5, 1.0);

Swizzling:

- float2 f2 =f3Normale.xy; // vec2 = (4.0, -2:0)
- float frY = f2UV.y; // frY = 1.0
- float3 frColor3 = frY.xxx; // frY = (1.0,1.0,1.0)

Möglich sind die Suffixe x,y,z,w oder r,g,b,a

Schreibmaskierung

Vorgefertigte Konstruktoren, z.B::

- float4 f4 = float3(1.0, 3.0, 2.0, 0.5);
- float3 f3Normale1 = float3(4.0, -2.0, 5.0);
- float2 f2UV = float2(0.5, 1.0);

Schreibmaskierung (Write masking):

- float vec4.xw = 7.0; // vec1 = (7.0, 3.0, 2.0, 7.0)

Möglich sind auch hier die Suffixe x,y,z,w oder r,g,b,a

Discards

- Durch das vorgefertigte Wort discard wird die Routine abgebrochen. Anders als bei return, welche Cg/HLSL ebenfalls bereitstellt, werden alle bisher berechneten Ergebnisse verworfen.

Hardware-Limitierungen

- Cg/HLSL-Programme werden sehr stark von der GPU- Hardware limitiert.
- Die Limitierungen variieren stark von Grafikkarte zu Grafikkarte
- Es gibt hardwareseitige Limitierungen der
  - Registergrößen
  - Arraygrößen
  - Instruktionenanzahl

Übung zu StyleOwn

Erzeugen Sie einen Rotstichfilter!

1

Freiwillige Zusatzaufgabe für die schnellen Nerds:

2

Erzeugen Sie ihren eigenen Toon-Shader!

3

Freiwillige Zusatzaufgabe für die Hardcore-Nerds:

4

Erzeugen Sie ein stereoskopisches Bild für die Rot-Grün-Brille!

5

GAME OVER