Masivně paralelní zpracování obrazu v prostředí systému VisionLab

25. 9. 2013 Liberec

Roman Cagaš, rc@mii.cz

Moravské přístroje a.s. - oblasti vývoje a výroby

Prostředí pro vývoj aplikací

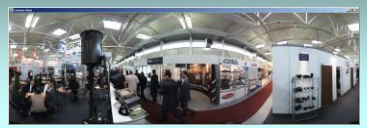
Control Web®

Software pro strojové vidění

VisiOn Lab™

Software pro vědecké kamery

Speciální a zakázkový software



moravské přístroje®

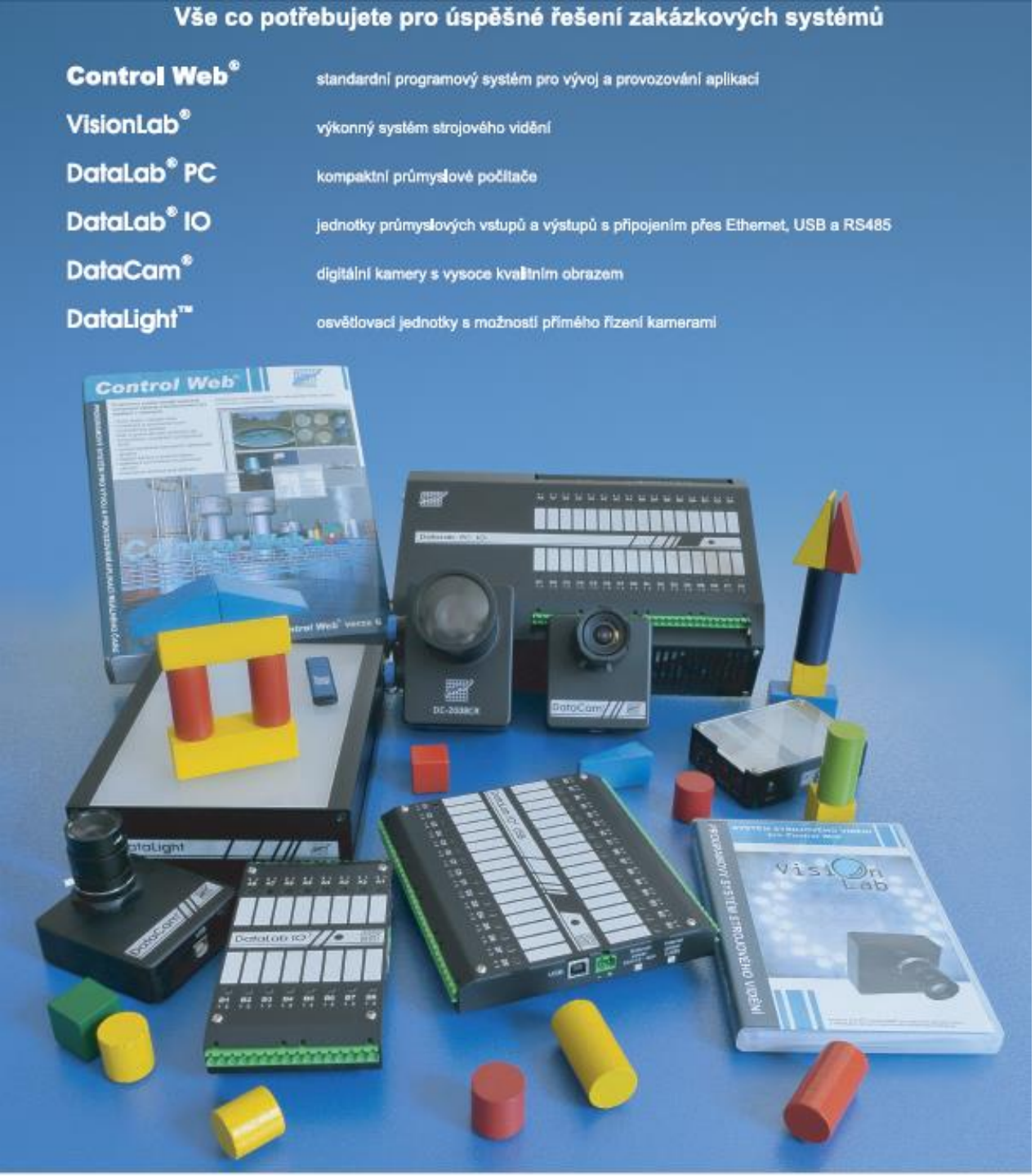
DataLab

Speciální technika

DataCam

DataLight

Vědecké kamery



Ekosystém produktů

- Velmi efektivní systém pro produkci malosériových a kusových zařízení
- Jednotné programové prostředí
- Distribuovaný systém v síťovém prostředí
- Databázová konektivita, webové technologie, vizualizace, virtuální realita, strojové vidění, přímé řízení strojů, OPC a mnoho dalšího …

DataCam®

- RAW data digitální kamery
- Vysoce kvalitní obraz
- Standardní i odolné provedení
- Připojení přes USB 2.0
- Objektivy standardů C, CS, bajonet Canon, bajonet Nikon

DataCam® - parametry

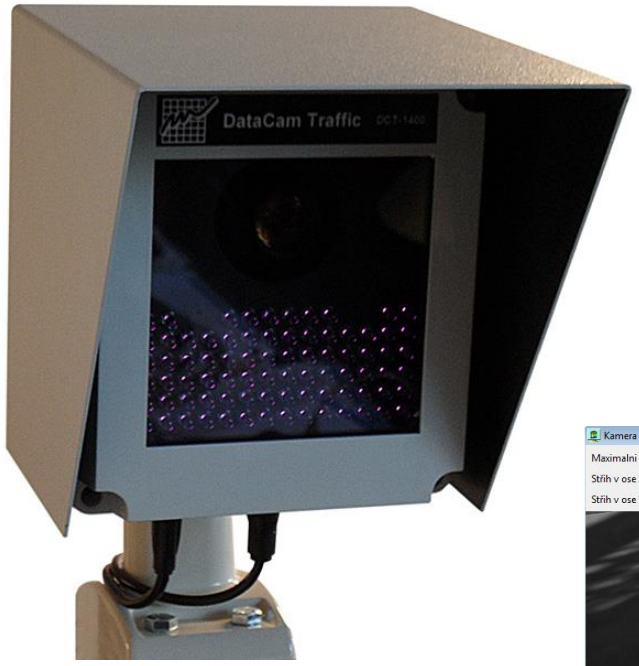
Kamery s 8-bitovou digitalizací obrazu se vyznačují vysokou rychlostí přenosu syrových obrazových dat do počítače při zachování nízkého šumu a vysoké přesnosti a stability obrazu.

	DC-0308	DC-0308C	DC-0808	DC-0808C	DC-1408	DC-1408C	DC-2008	DC-2008C
CCD čip	Sony ICX424AL	Sony ICX424AQ	Sony ICX204AL	Sony ICX204AK	Sony ICX285AL	Sony ICX285AQ	Sony ICX274AL	Sony ICX274AQ
Velikost čipu	1/3“	1/3“	1/3“	1/3“	2/3“	2/3“	1/2“	1/2“
Rozlišení	640 x 480	640 x 480	1024 x 768	1024 x 768	1392 x 1040	1392 x 1040	1600 x 1200	1600 x 1200
Velikost pixelu	7,4 µm	7,4 µm	4,65 µm	4,65 µm	6,45 µm	6,45 µm	4,4 µm	4,4 µm
Obrazová plocha	4,9 x 3,7 mm	4,9 x 3,7 mm	4,8 x 3,6 mm	4,8 x 3,6 mm	9,0 x 6,7 mm	9,0 x 6,7 mm	7,2 x 5,4 mm	7,2 x 5,4 mm
Rozlišení převodníku	8 bitů	8 bitů	8 bitů	8 bitů	8 bitů	8 bitů	8 bitů	8 bitů
Barevnost	monochrom	RGGB mozaika	monochrom	RGGB mozaika	monochrom	RGGB mozaika	monochrom	RGGB mozaika
Snímková frekvence	48 FPS	48 FPS	20 FPS	20 FPS	11 FPS	11 FPS	8 FPS	8 FPS

Kamery s 16-bitovou digitalizací obrazu se vyznačují velmi vysokou dynamikou a kvalitou obrazu. Hodí se pro aplikace s nízkou úrovní osvětlení a vysokými požadavky na přesnost a reprodukovatelnost měření.

	DC-0316	DC-0316C	DC-0816	DC-0816C	DC-1416	DC-1416C	DC-2016	DC-2016C
CCD čip	Sony ICX424AL	Sony ICX424AQ	Sony ICX204AL	Sony ICX204AK	Sony ICX285AL	Sony ICX285AQ	Sony ICX274AL	Sony ICX274AQ
Velikost čipu	1/3“	1/3“	1/3“	1/3“	2/3“	2/3“	1/2“	1/2“
Rozlišení	640 x 480	640 x 480	1024 x 768	1024 x 768	1392 x 1040	1392 x 1040	1600 x 1200	1600 x 1200
Velikost pixelu	7,4 µm	7,4 µm	4,65 µm	4,65 µm	6,45 µm	6,45 µm	4,4 µm	4,4 µm
Obrazová plocha	4,9 x 3,7 mm	4,9 x 3,7 mm	4,8 x 3,6 mm	4,8 x 3,6 mm	9,0 x 6,7 mm	9,0 x 6,7 mm	7,2 x 5,4 mm	7,2 x 5,4 mm
Rozlišení převodníku	16 bitů	16 bitů	16 bitů	16 bitů	16 bitů	16 bitů	16 bitů	16 bitů
Barevnost	monochrom	RGGB mozaika	monochrom	RGGB mozaika	monochrom	RGGB mozaika	monochrom	RGGB mozaika
Snímková frekvence	16 FPS	16 FPS	6 FPS	6 FPS	3,5 FPS	3,5 FPS	2,5 FPS	2,5 FPS

DataCam® Traffic



- Kamera v klimaticky odolném provedení
- Vysoká citlivost v blízkém IR spektru
- Vestavěný IR osvětlovač

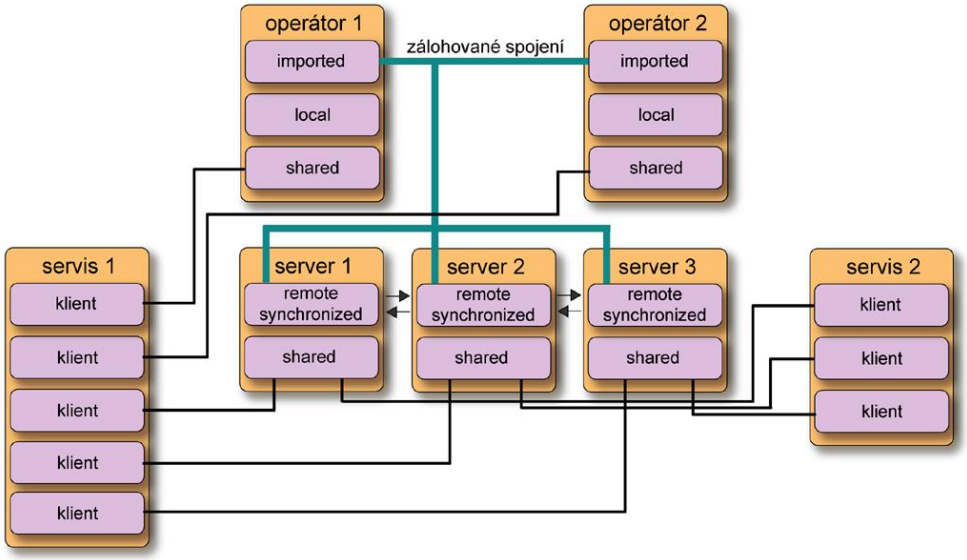
Funkce čtení registračních značek automobilů je snadno k dispozici v podobě jediného kroku strojového vidění systému VisionLab

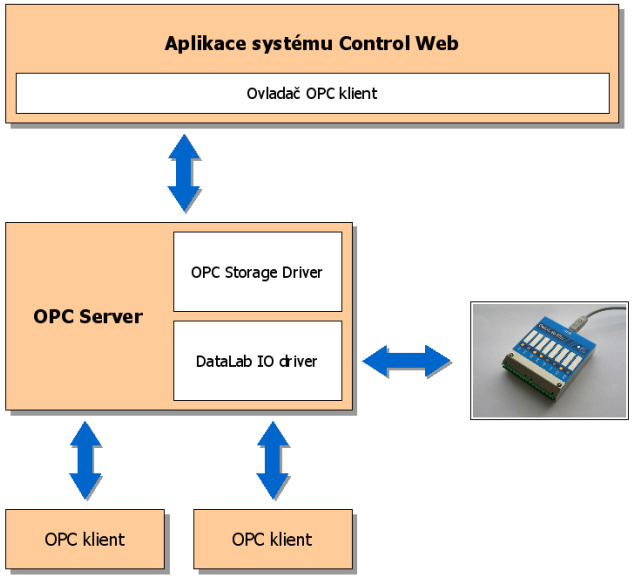
DataLight®

- Diodové osvětlovací jednotky přímo řiditelné kamerami
- Zadní osvětlovače, kruhové souosé osvětlovače, reflektory, osvětlovače v temném poli
- Volitelná barva světla, směrovost a typy difuzorů
- Osvětlovače s trvalým svitem i záblesková

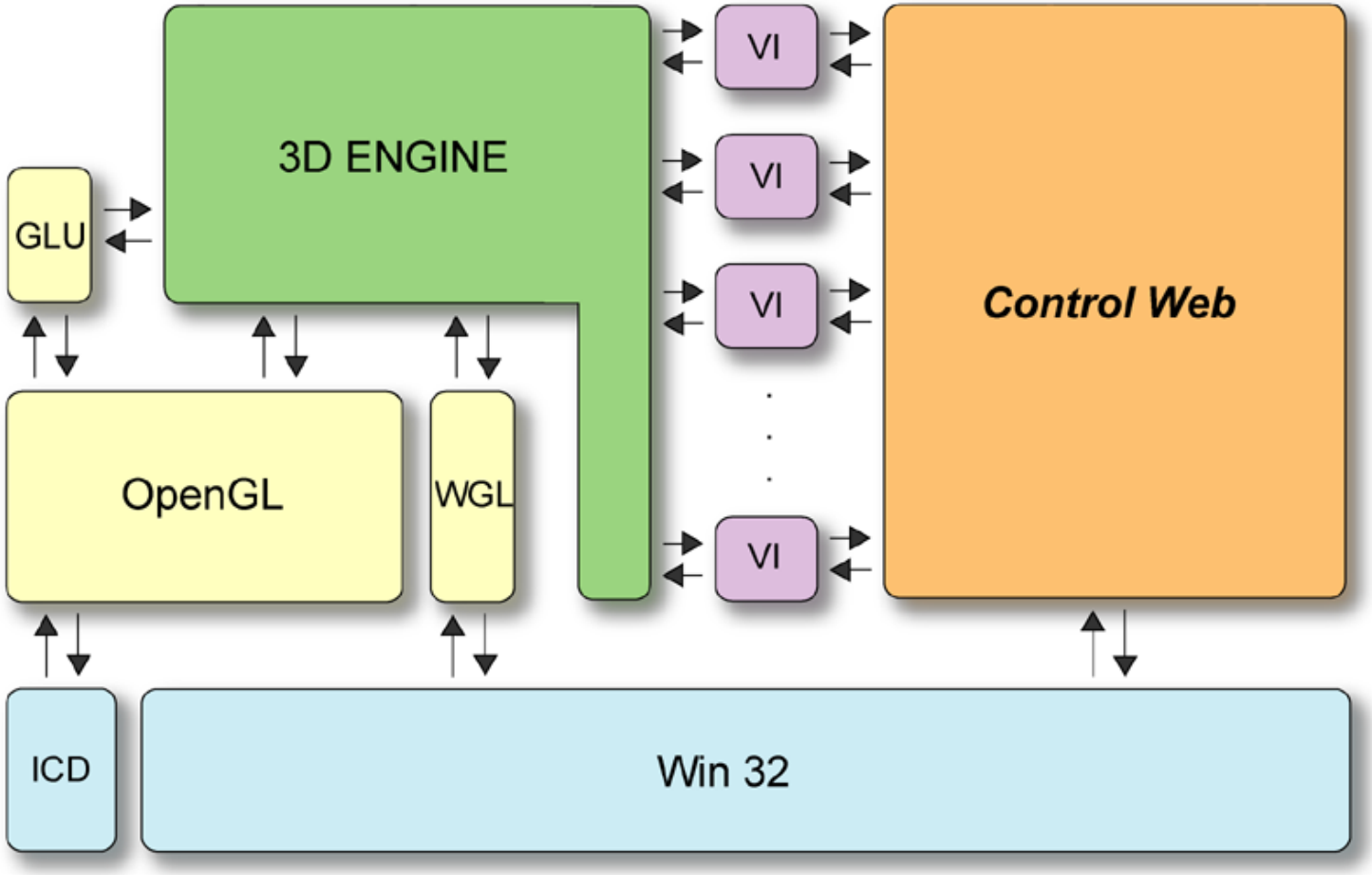
Control Web
komponenty

- Komponenty virtuálních přístrojů
- Komponenty ovladačů
  - S vlastnických protokoly
  - Se standardními protokoly
    - Modbus, Modbus TCP
    - SMS
    - HTTP klient
    - Modem, radiomodem
  - Pro standardní sw API
    - OPC server, klient
  - Univerzální ovladače
    - TCPIP packet
    - Serial text
- Komponenty kroků strojového vidění
- Komunikace mezi jádry systému
  - Sdílení a synchronizace dat
- SQL, ODBC
- OCL – scripting nad komponentami
- Multiprocessing
- Masivně paralelní GPU processing
- Atd. atp. …



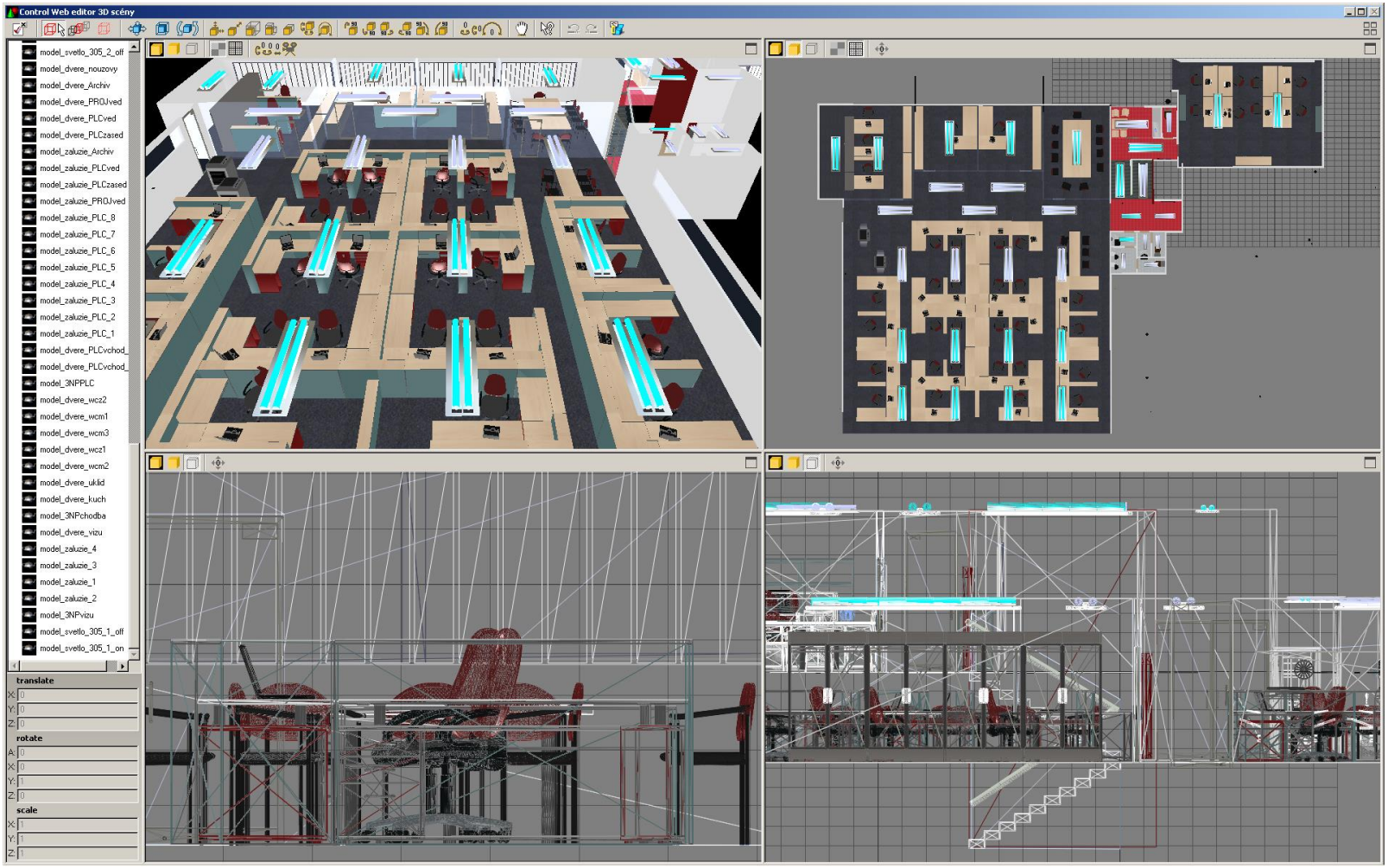


Control Web – 3D vykreslovací systém



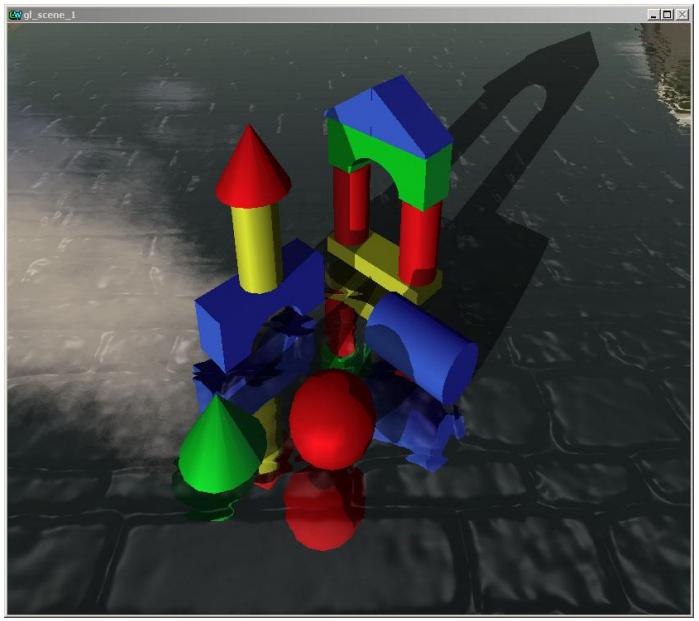
- Celý systém je dynamicky zaváděn až v případě jeho potřeby
- Architektura klient – server
- Vykreslovací systém v separátním threadu – minimální ovlivňování běhu aplikačního programu
- OpenGL + GLSL shadery

Control Web – editace 3D scény

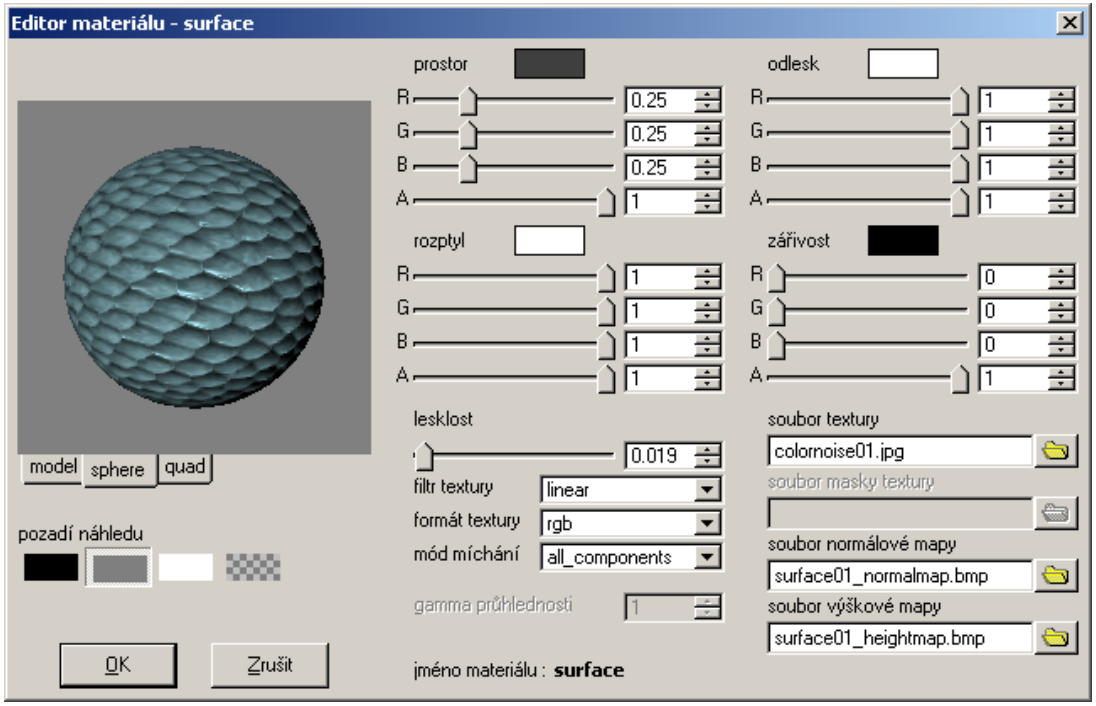


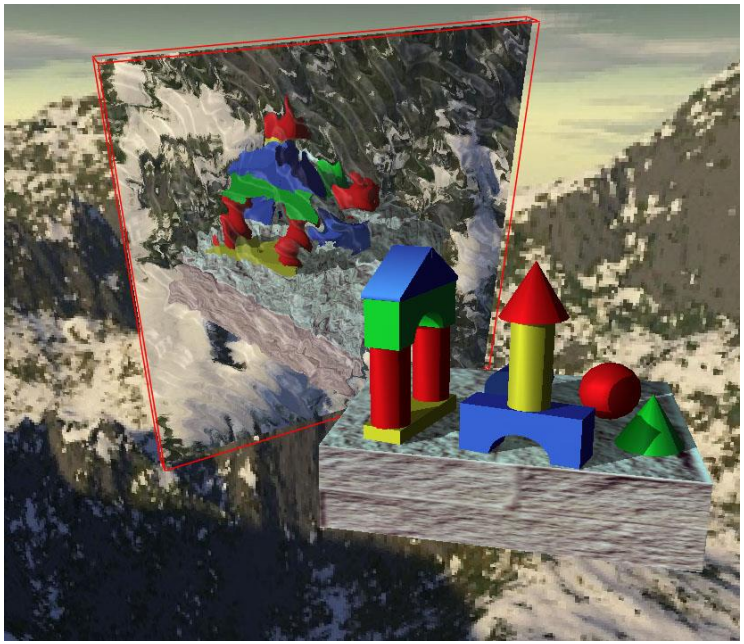
- In-place editor
- Samostatný editor
- Import 3D modelů ve formátech OBJ, 3DS a DXF
- Postprocessing obrazu atd. atp.

Control Web – vykreslovací server

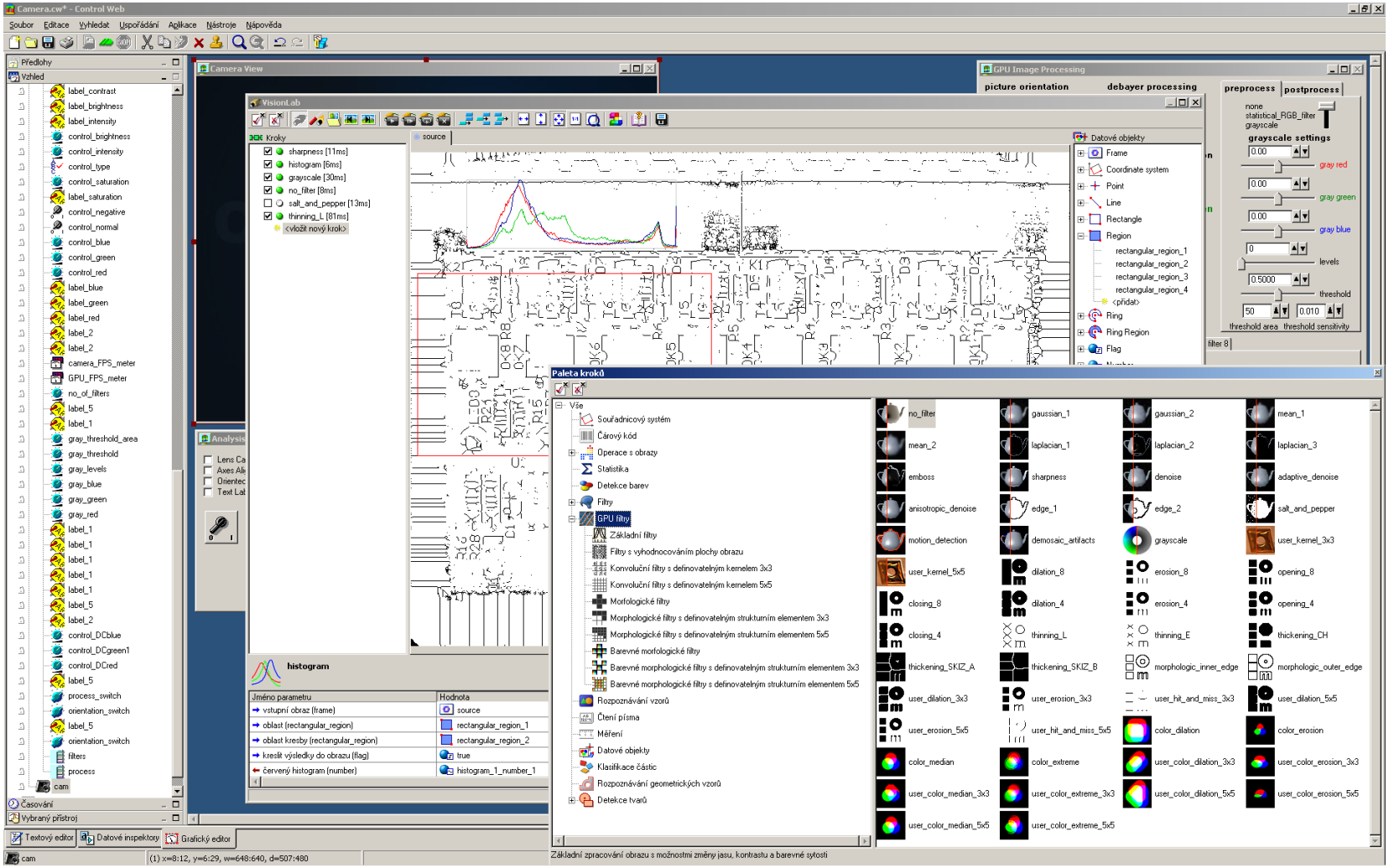


bumpmapy, stíny, zrcadlení, mlha, environment, postprocessing …





VisionLab – systém strojového vidění



- Sada komponent instalovatelná do prostředí systému **Control Web**
- Grafický editor nad kamerovými virtuálními přístroji
- Algoritmy strojového vidění jsou zařazeny do aplikačního programu v prostředí systému **Control Web**

GPU systémech strojového vidění

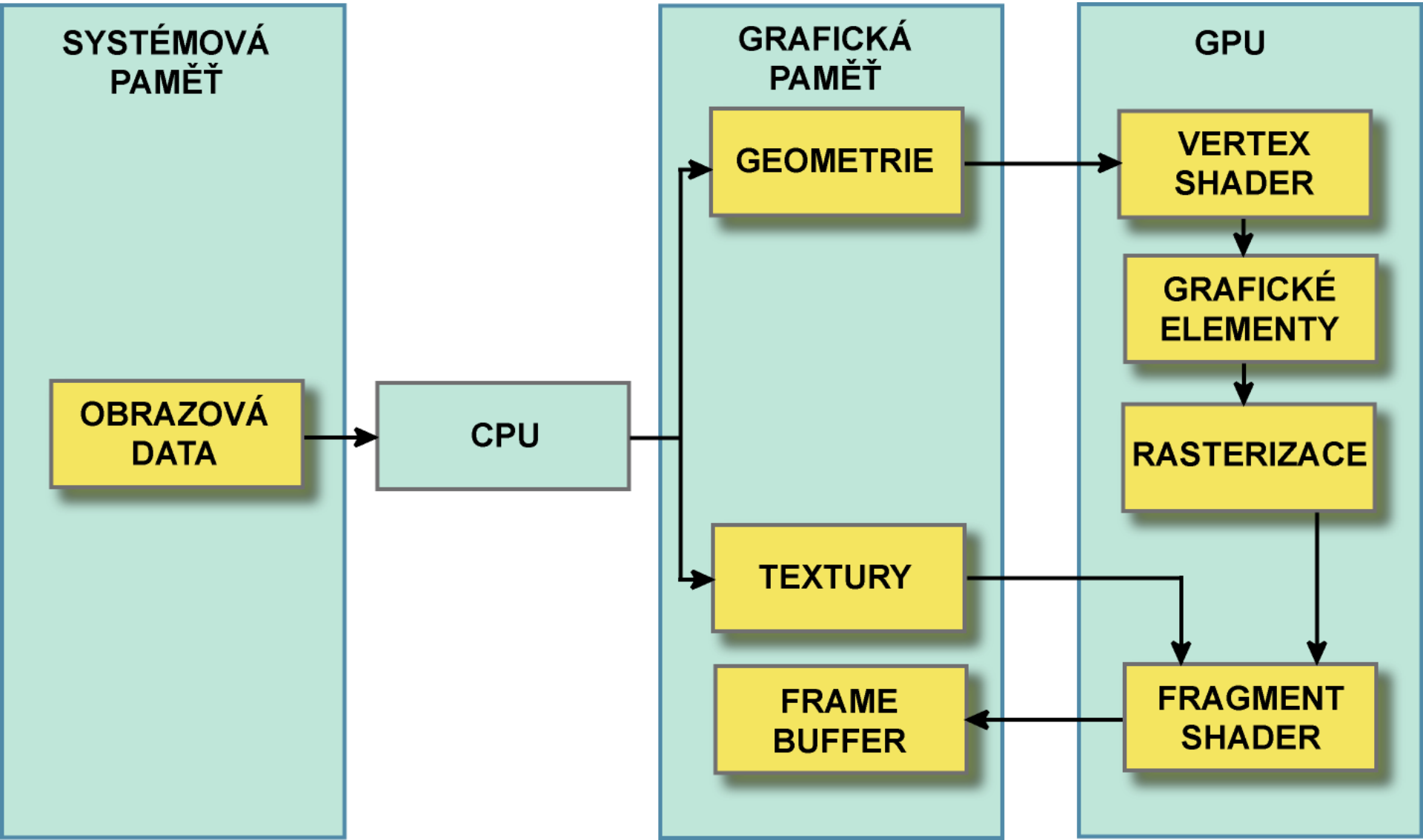
- GPU je samozřejmou součástí veškerých současných počítačů, tabletů, telefonů ...
- CPU -> jednotky výpočetních jader
- GPU -> stovky jader, široké sběrnice, vysoká datová propustnost
- VisionLab – automatický multithreading na více jádrech CPU + kroky v GPU

např. GeForce GTX 680

• Počet tranzistorů	3.5 miliardy
• Počet výpočetních jader	1536
• Hodinové frekvence GPU	1006 MHz
• Výkon přístupu k texturám	128.8 miliard pixelů / sec
• Šířka sběrnice grafické paměti	384 bitů
• Datová propustnost	192.2 GB / sec
• Výpočetní výkon (single precision)	3.79 TFLOPS

Není škoda to nevyužít?

Tradiční architektura zpracování grafických dat



Jazyky pro programování shaderů

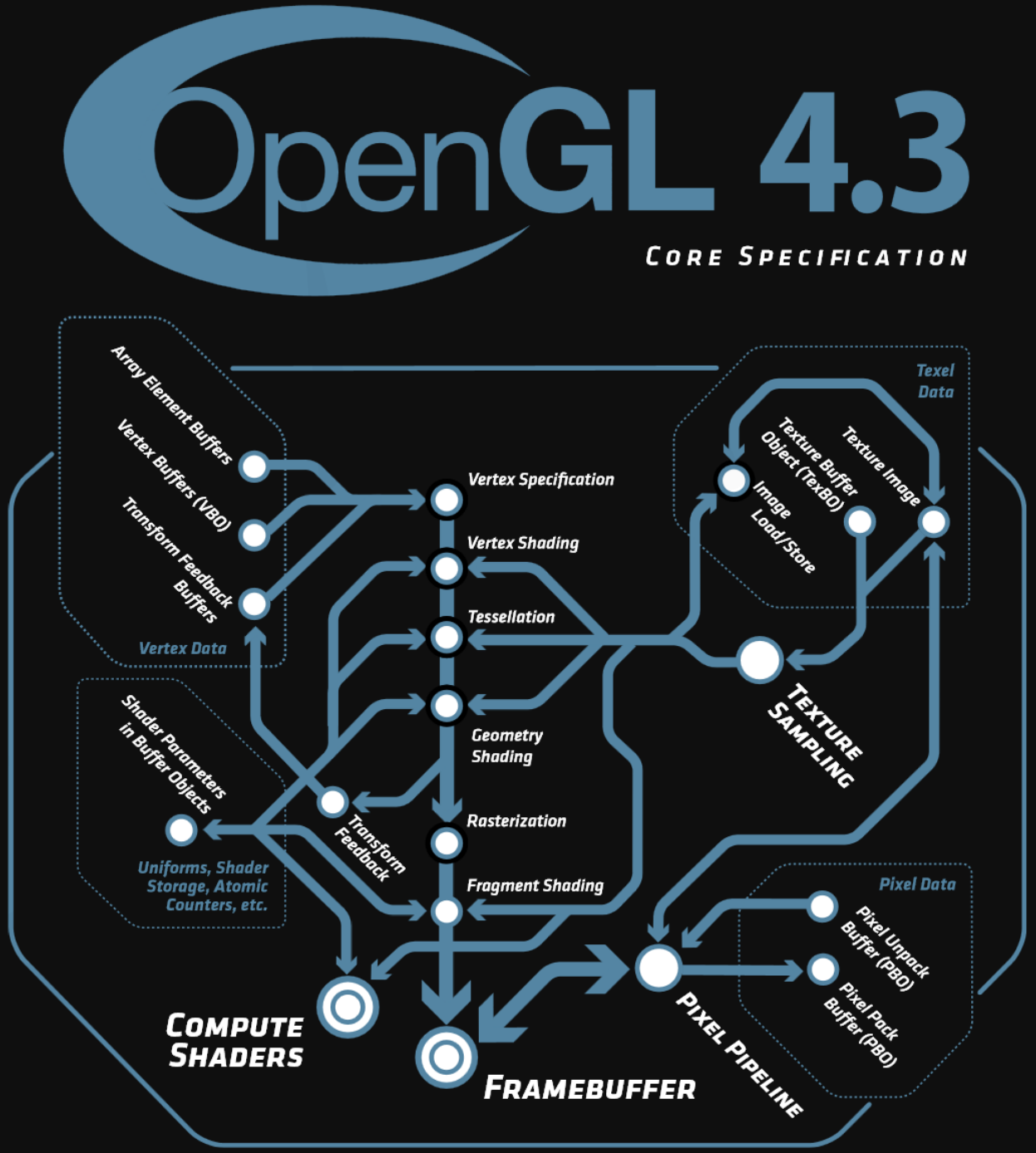
- Cg (NVIDIA pro DirectX i OpenGL)
- HLSL „High-Level Shading Language“ (Microsoft DirectX)
- GLSL „OpenGL Shading Language“

Prostředí pro GPU computing

- CUDA (NVIDIA)
- OpenCL
- OpenGL + rozšíření pro pro sdílení a přenos bloků dat

OpenGL

pravděpodobně nejsilnější standard do budoucna



Prostředky GPU

- **Programovatelné procesory** – v uniformní architektuře mohou vystupovat jako vertexové, fragmentové a geometrické. Při zpracování obrazu z kamery využijeme tradiční funkce vertexových a fragmentových shaderů:

  Vertexový shader je spuštěn pro každý vertex, vykreslujeme-li např. trojúhelník, je spušten celkem třikrát. Jeho výsledkem je grafický element, který je předán do rasterizeru.

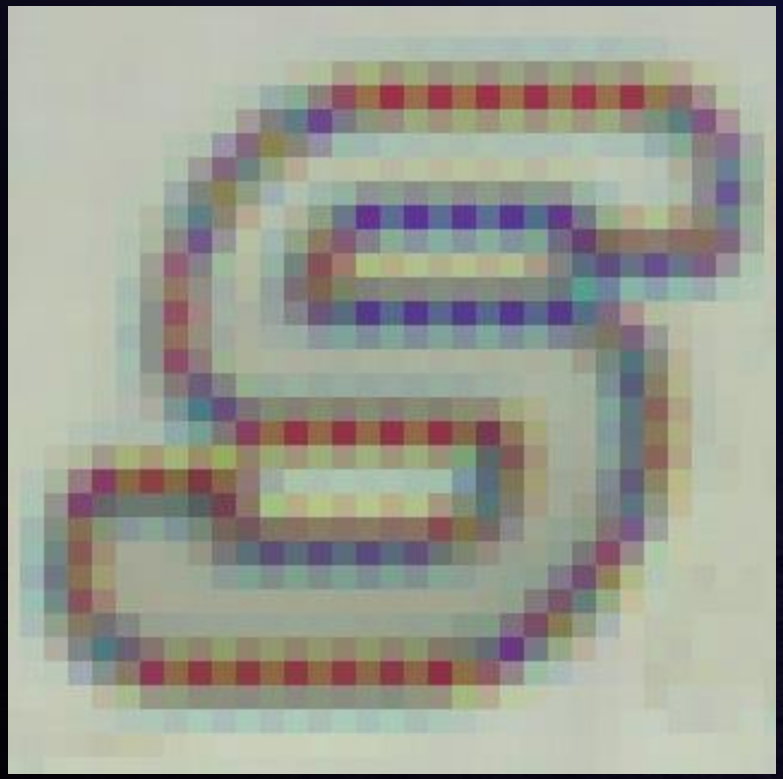
  Fragmentový shader je spoušten výstupem rasterizeru pro každý fragment, tedy pro každý obrazový bod vykreslované scény.
- **Rasterizer** – prostřednictvím interpolace vytváří jednotlivé fragmenty z obrazových elementů. Základním obrazovým elementem je trojúhelník, definovaný třemi vertexy. V jeho ploše jsou pak rasterizerem interpolovány texturové souřadnice nebo barvy fragmentů.
- **Textury** – jsou bloky grafické paměti, které mohou být čteny fragmentovými procesory. Data z texturových objektů lze pouze číst, zápis do nich není fragmentovými shadery možný.
- **Framebuffer** – je blok paměti, do které fragmentové procesory zapisují výsledku běhu fragmentových shaderů. Data uvnitř framebufferu nelze fragmentovým procesorem číst a dokonce každý s paralelně běžících fragmentových shaderů může zapsat jen do jediné pozice v bufferu, která odpovídá jeho instanci v rámci plochy obrazu.

Využití GPU v systému strojového vidění VisionLab

- GPU umožňuje realizovat takové algoritmy, které by jinak byly v akceptovatelném čase nerealizovatelné
- Virtuální přístroj **gl_camera**:
  - Využívá vykreslovací stroj systému *Control Web*
  - Vykreslovací stroj pracuje ve vlastním threadu
  - Vykreslovací stroj používá architekturou klient-server

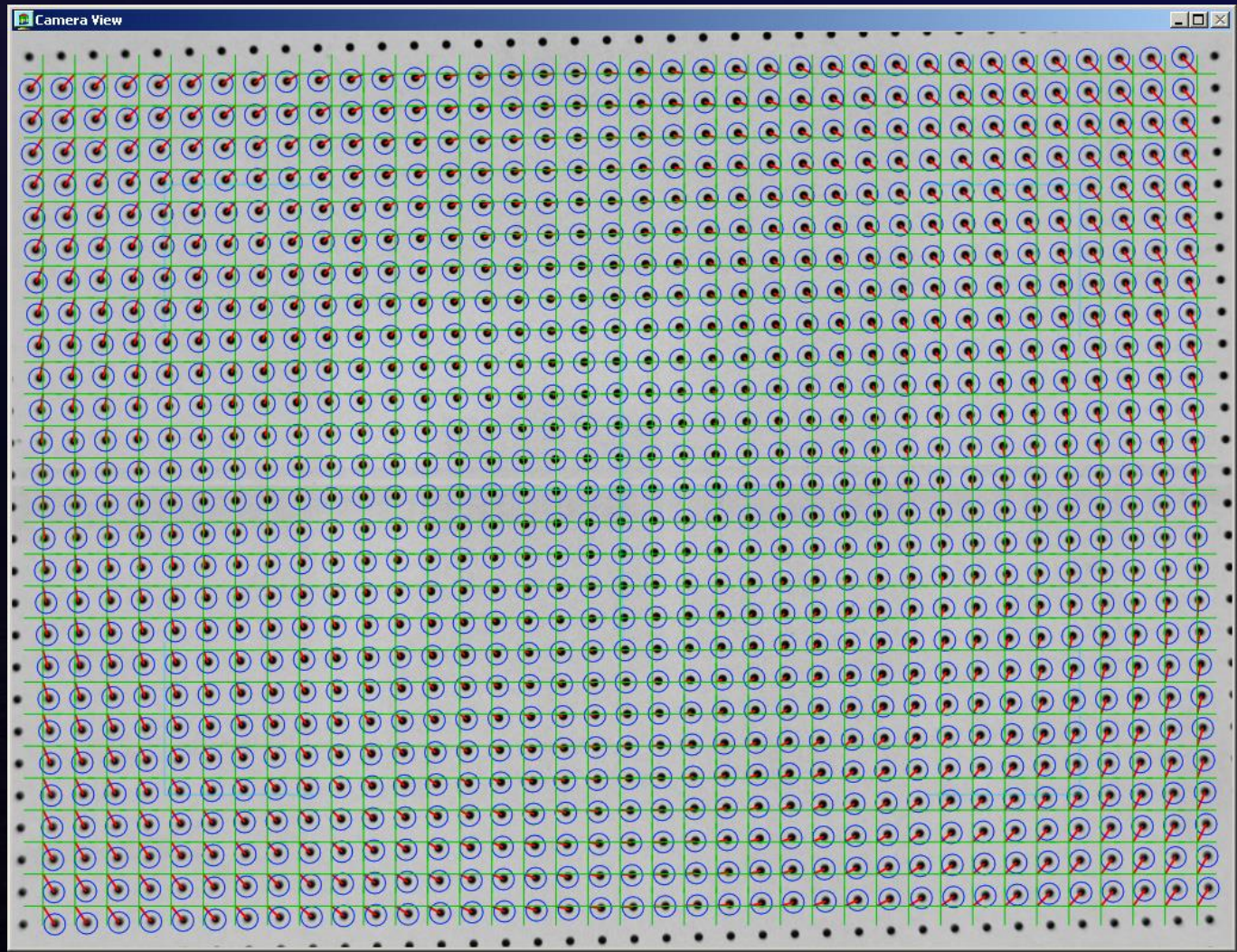
Pokročilá adaptivní interpolace barevné mozaiky

- Víceprůchodový algoritmus s float-point výpočty

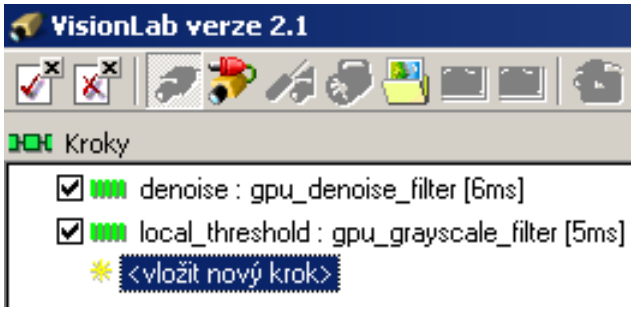


Korekce geometrických zkreslení obrazu

- tvorba výsledného obrazu řešena s vysokou subpixelovou přesností programem fragmentového shaderu, který poskytuje zaručeně kvalitní a stabilní výsledky na veškerých GPU



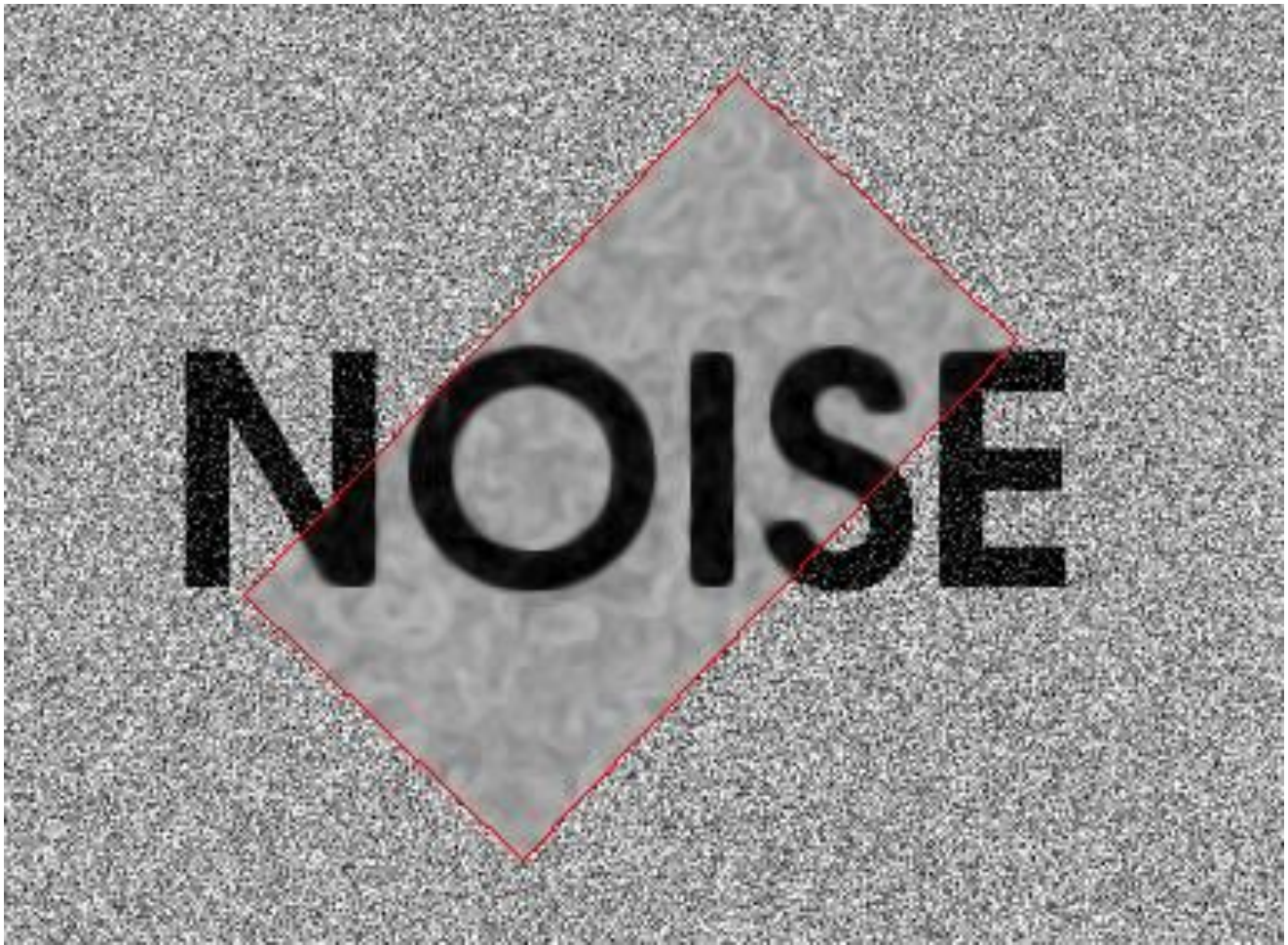
Obrazové filtry



- Obrazové filtry vypadají na první pohled jako optimální úlohy pro řešení v GPU. Často tomu tak je, ale nemusí to platit univerzálně. Potíž může být v tom, že řada jednoduchých kernelových filtrů nemá příliš vysokou výpočetní intenzitu (a současné vícejádrové CPU také nepočítají až tak zoufale pomalu). Režie přenosu obrazových dat mezi systémovou a grafickou pamětí pak může i stonásobně vyšší rychlost výpočtů kernelu v GPU oproti CPU znehodnotit.
- Situace se ale dramaticky změní v případě filtrů s nutností složitějších výpočtů v plovoucí řádové čárce, jako je tomu např. u transformací barevných prostorů, řešení saturační matic, šumových filtrů atd. Pak nám může využití GPU zrychlit tyto kroky až o dva řády.

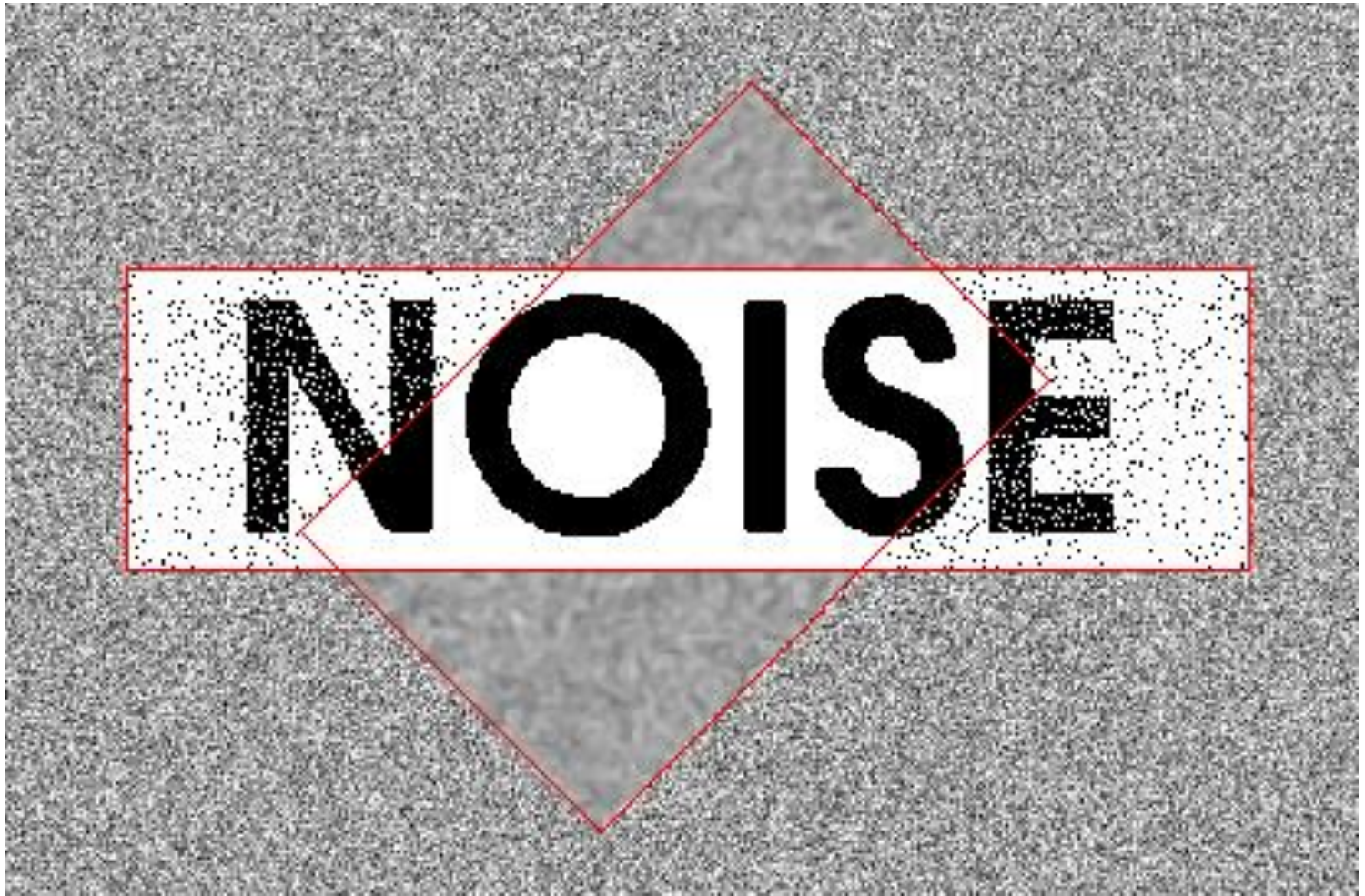
Obrazové filtry

- Anisotropický šumový filtr využívá výkonu GPU ve výpočtech s plovoucí řádovou čárkou



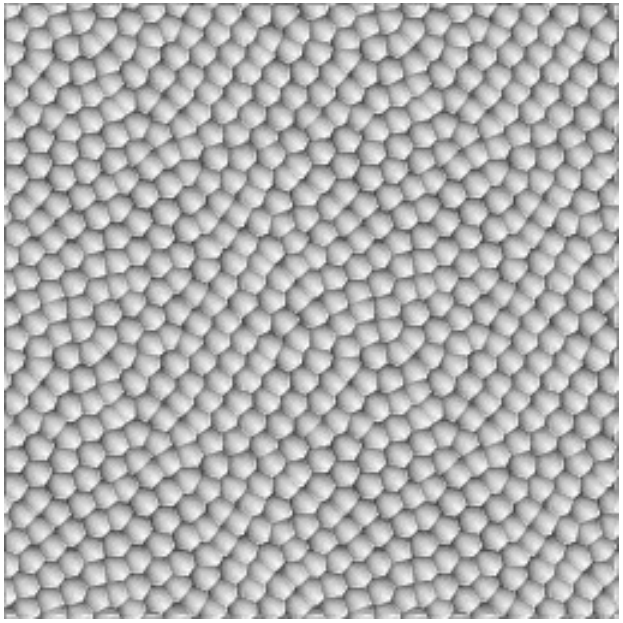
Obrazové filtry

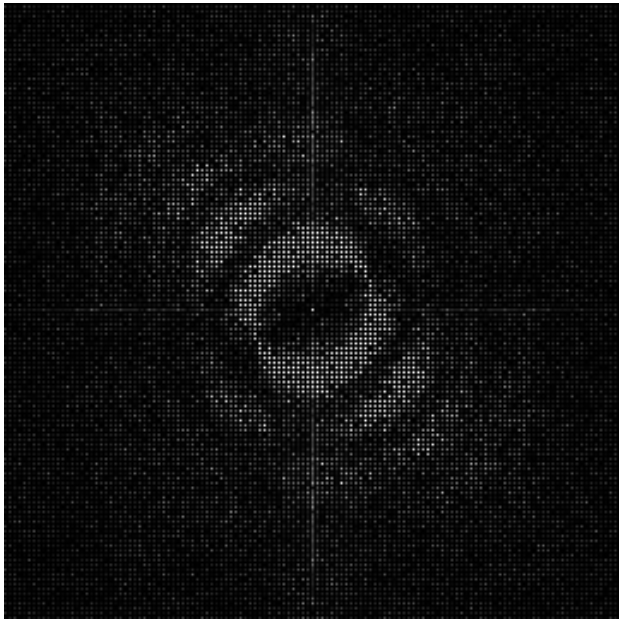
- Lokálně prahovací filtr využívá mohutného datového toku GPU při čtení pixelů textur



Transformační kódování

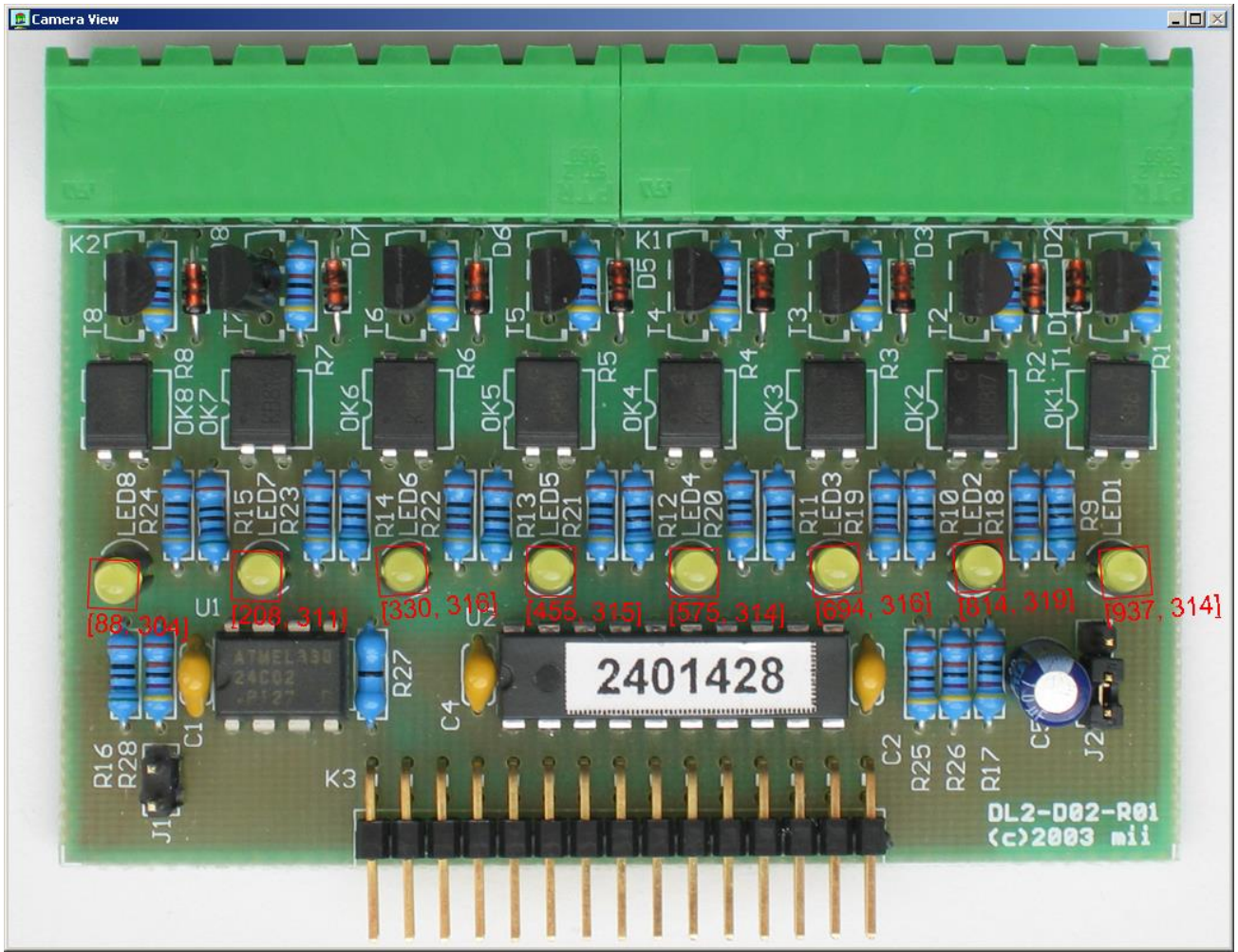
- Dvojrozměrná Fourierova transformace obrazu je natolik výpočetně intenzivní, že v reálném čase není jinak než s využitím GPU realizovatelná





Další možnosti

- Existují však i problematické úlohy, kde přínosy GPU nemusejí být takto jasné.
- Příkladem je např. hledání vzorů realizované prostřednictvím GPU - tento příklad trvá asi 50ms na procesoru G80

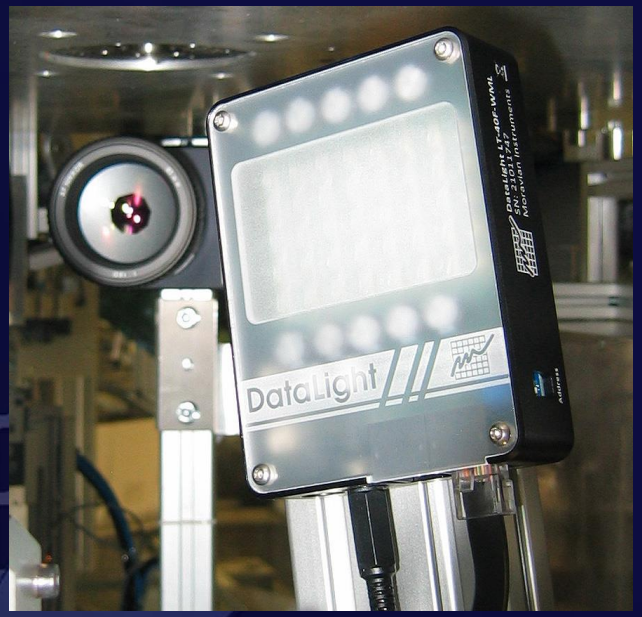


API pro kroky systému VisionLab

- Systém strojového vidění VisionLab poskytuje krokům, které využívají GPU značný prostor. S vykreslovacím jádrem systému Control Web i virtuálním přístrojem gl_camera komunikují jen prostřednictvím velmi jednoduchého povelového rozhraní. Kroky dostanou k dispozici paměťový pixel buffer dané velikosti a formátu pixelů a mohou si zcela samostatně alokovat libovolné prostředky GPU – mohou si vytvářet např. vlastní texturové objekty, další frame buffer objekty, samozřejmě mohou načítat a spouštět vlastní shadery atd. Kroky strojového vidění tak mohou využívat veškerá nejnovější rozšíření, které přinášení aktuální grafické ovladače. Tyto možnosti umožňují systému strojového vidění VisionLab stále sledovat poslední vývoj technologií a přinášet tak svým uživatelům vysoký užitek a konkurenční výhodu.

Příklad použití příkazového API při zpracování obrazu v kroku:

```
 // begin rendering
  if (! cw_iproc::CameraCallback(CallbackHandle, vgltype::commandHIGH, (WCHAR  *)& vgltype::commandLockRenderingContextW, sizeof(
vgltype::TGPUCommandData ), (ADDRESS)& GPUCommandData ) ) {
    wcsncpy( ErrorString, GPUCommandData.ErrorString.a_, ErrorString_HIGH + 1);
    ok = false;
  }
  // initialize rendering
  if (! cw_iproc::CameraCallback( CallbackHandle, vgltype::commandHIGH, (WCHAR  *)& vgltype::commandInitializeRenderingW, sizeof(
vgltype::TGPUCommandData ), (ADDRESS)& GPUCommandData ) ) {
    wcsncpy( ErrorString, GPUCommandData.ErrorString.a_, ErrorString_HIGH + 1);
    ok = false;
  }
 // rendering
  LoadTextureFromBuffer( SourceTexture, GL_TEXTURE0, w, d, bpp, data_addr );
  SetSourceDimensions( w, d, bpp );
  Render( PRegion );
  // copy rendered picture to output frame
  if (! cw_iproc::CameraCallback( CallbackHandle, vgltype::commandHIGH, (WCHAR  *)& vgltype::commandPictureDataToOutputFrameW, sizeof(
vgltype::TGPUCommandData ), (ADDRESS)& GPUCommandData ) ) {
    wcsncpy( ErrorString, GPUCommandData.ErrorString.a_, ErrorString_HIGH + 1);
    ok = false;
  }
  // finalize rendering
  if (! cw_iproc::CameraCallback( CallbackHandle, vgltype::commandHIGH, (WCHAR  *)& vgltype::commandFinalizeRenderingW, sizeof(
vgltype::TGPUCommandData ), (ADDRESS)& GPUCommandData ) ) {
    wcsncpy( ErrorString, GPUCommandData.ErrorString.a_, ErrorString_HIGH + 1);
    ok = false;
  }
  // end rendering
  if (! cw_iproc::CameraCallback( CallbackHandle, vgltype::commandHIGH, (WCHAR  *)& vgltype::commandUnlockRenderingContextW, sizeof(
vgltype::TGPUCommandData ), (ADDRESS)& GPUCommandData ) ) {
    wcsncpy( ErrorString, GPUCommandData.ErrorString.a_, ErrorString_HIGH + 1);
    ok = false;
  }
```



Děkuji!

Ať vám vaše systémy strojového vidění spolehlivě fungují a ať se při jejich vývoji příliš nenadřete.