ŠIAULIŲ UNIVERSITETAS
TECHNOLOGIJOS FAKULTETAS
RADIOTECHNIKOS KATEDRA

Donatas Rimkus

Srautinių komandų taikymas videookulografijoje

Magistro darbas

Darbo vadovas
prof. V. Laurutis

Darbo konsultantas
doc. G. Daunys

Darbo recenzentas
E. Paliulis

Šiauliai
2005

Turinys

Įvadas

Žmogaus akys nuolatos juda, netgi ir tada, kai mėginama pastoviai fiksuoti nejudantį tašką. Žinoma, kad fiksacijos metu judėjimo amplitudė yra maža. Todėl toks akių judėjimas vadinamas mikrojudėjimu. Neatsižvelgiant į mažas amplitudes, akių mikrojudesiai gali padėti ištirti žmogiškus atpažinimo procesus ir diagnozuoti akių sistemos ligas. Akių judesių sekimas realiame laike gali būti panaudojamas palengvinant neigalių žmonių darbui (pavyzdžiui darbui su kompiuteriu).

Akių judesius galima rasti pagal vyzdžio centro koordinates, kurias nustatyti galima naudojant tam tikrus metodus. Vienas iš šių metodų yra videookulografinis metodas. Videookulografiniame metode yra naudojama video kamera, todėl gaunamas didelis duomenų kiekis. Kad pritaikyti videookulografinį metodą realiame laike, reikalinga didelė duomenų apdorojimo sparta. Duomenų apdorojimo pagreitinimui galima naudoti srautines komandas.

Šiame darbe buvo siekiama ištirti duomenų apdorojimo pagreitėjimą panaudojus stautines komandas. Todėl šio tyrimo tikslas – pagreitinti vyzdžio centro koordinačių nustatymo metodų skaičiavimus.

1. Vyzdžio centro koordinačių nustatymo metodika

Vienas pagrindinių uždavinių, videookulografijoje, yra pagreitinti vyzdžio centro koordinačių metodų skaičiavimus. Šiame darbe yra išanalizuoti du vyzdžio centro koordinačių nustatymo metodai: analizė tiesėmis ir analizė apskritimu. Šie metodai įgyvendinti dviemis programinėmis įrangomis: viena be srautinių komandų, o kita su srautinėmis komandomis. Dvi programinės įrangos rašomos todėl, kad vėliau būtų galima palyginti jų gautus rezultatus.

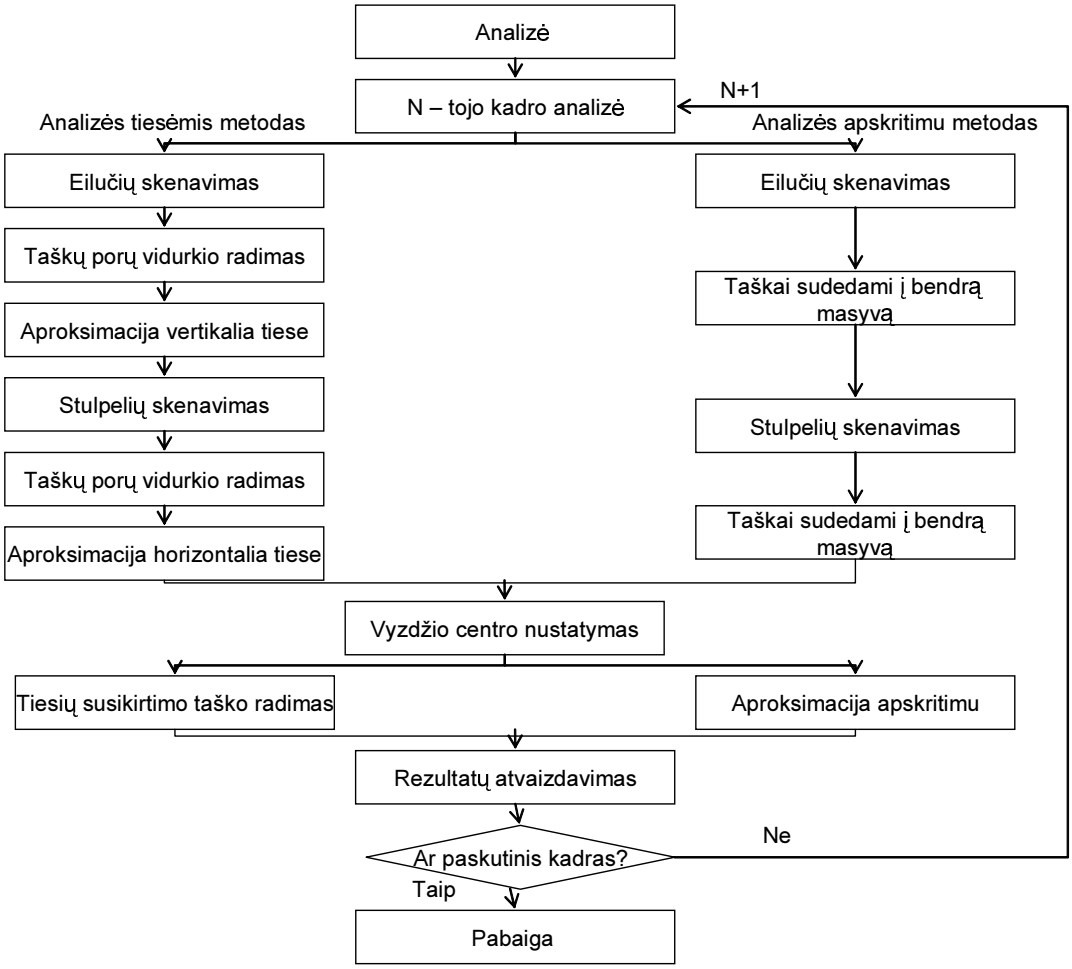
1.1. Vyzdžio centro koordinačių nustatymo algoritmai

Vyzdžio centro koordinačių metodų analizės algoritmas yra pateiktas 1 paveiksle. Analizės tiesėmis metodo algoritmas susideda iš 5 žingsnių:

1. eilučių skenavimas;
2. stulpelių skenavimas;
3. aproksimacija vertikalia tiese;
4. aproksimacija horizontalia tiese;
5. vyzdžio centro koordinačių nustatymas.

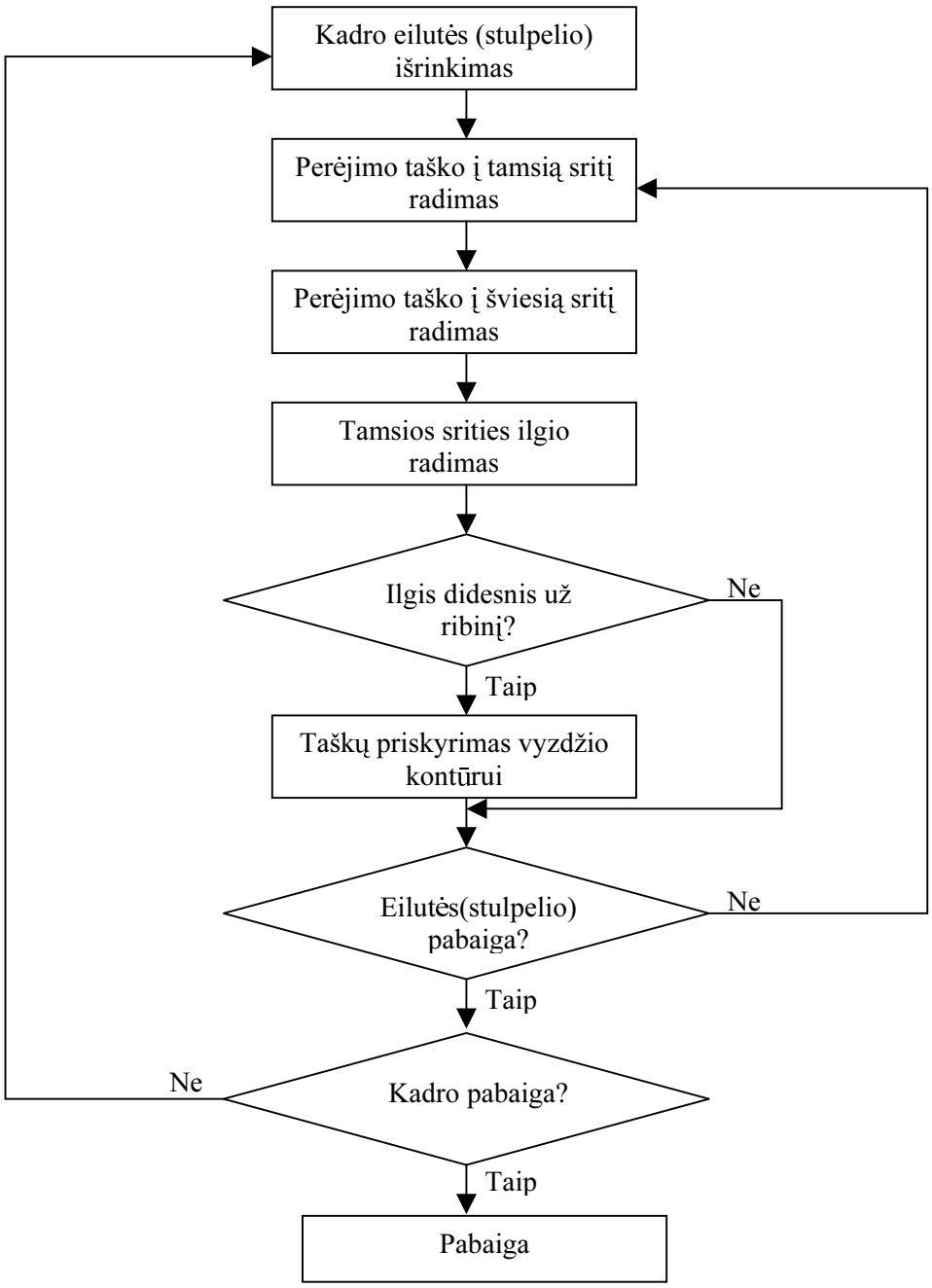
Analizės apskritimu metodo algoritmas susideda iš 3 žingsnių:

1. eilučių skenavimas;
2. stulpelių skenavimas;
3. vyzdžio centro koordinačių nustatymas.



1 pav. Vyzdžio centro koordinačių aptikimo algoritmas [2]

Eilučių (stulpelių) skenavimo algoritmas yra pateiktas 2 paveiksle. Atlikus šį algoritmą yra aptinkamas vyzdžio konturas. Visi gauti taškai, kurie yra priskiriami vyzdžio konturui, surašomi į masyvą.



2 pav. Vyzdžio kontūro taškų nustatymo skenavimo metodo algoritmas [2]

1.2. Analizės tiesėmis metodas

Kaip jau buvo minėta, analizė tiesėmis metodas yra atliekamas 5 žingsniais. Po 1 ir 2 žingsnio (eilučių skenavimo ir stulpelių skenavimo) yra randamas pilnas vyzdžio konturas.

Pirmajame žingsnyje yra skenuojama kiekviena eilutė nuo pradžios iki pabaigos. Randamas dviejų aptiktų kontūro taškų, priklausančių tai pačiai eilutei, vidurkis. Rasti vidurkio taškai yra aproksimuojami vertikalia linija, t.y. randama tiesės lygtis:

$$y = v_0 + v_1 x, \tag{1}$$

kur v_0 ir v_1 – koeficientai. Šie koeficientai apskaičiuojami remiantis mažiausių kvadratų metodu:

$$v_0 = \frac{\sum_{i=1}^{n} y_i * \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i y_i * \sum_{i=1}^{n} x_i}{n * \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}, \tag{2}$$

$$v_1 = \frac{n * \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i * \sum_{i=1}^{n} y_i}{n * \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}, \tag{3}$$

kur n – taškų skaičius po koordinačių vidurkinimo.

Antras žingsnis yra analogiškas pirmajam, tik vietoje eilučių yra skenuojami stulpeliai. Randamas dviejų aptiktų kontūro taškų, priklausančių tam pačiam stulpeliui, vidurkis. Aproksimuojant rastus vidurio taškus horizontalia linija, yra randama horizontalios tiesės lygtis:

$$y = h_0 + h_1 x; \tag{4}$$

Siekiant rasti vyzdžio centro koordinates (x_0, y_0) yra sprendžiama lygčių sistema, susidedanti iš 1 ir 4 lygčių [2]:

$$\begin{cases} y = v_0 + v_1 x, \\ y = h_0 + h_1 x; \end{cases} \tag{5}$$

tuomet:

$$x_0 = \frac{h_0 - v_0}{v_1 - h_1}; \tag{6}$$

$$y_0 = v_0 + v_1 x_0 . \tag{7}$$

1.3. Analizės apskritimu metodas

Analizė apskritimu metodas yra atliekamas 3 žingsniais. Po 1 ir 2 žingsnio (eilučių skenavimo ir stulpelių skenavimo), kaip ir analizėje tiesėmis metode yra randamas pilnas vyzdžio konturas. Po eilučių skenavimo, bei atitinkamai po stulpelių skenavimo rasti taškai, priklausantys vyzdžio konturui, yra surašomi į bendrą masyvą.

Trečiajame žingsnyje (aproksimacija apskritimu) yra randami vyzdžio parametrai (vyzdžio centro koordinatės (x_0, y_0) bei vyzdžio spindulys R). Siekiant tiksliai rasti šiuos parametrus yra apskaičiuojamos mažiausios kvadratinės paklaidos J, bei jos minimizuojamos:

$$J = \sum_{i=1}^{n} w_i \left[(x_i - x_0)^2 + (y_i - y_0)^2 - R^2 \right]^2 , \tag{8}$$

kur yra atliekamas visų rastų vyzdžio kontūro taškų (x_i, y_i) i = 1…n sumavimas, w_i – i-tojo taško svoris.

Minimizuoti galima, prilyginant J dalines išvestines pagal x_0, y_0 ir R iki nulio. Tuomet x_0, y_0 ir R išraiškos gali buti užrašytos taip:

$$\begin{aligned} x_0 &= \frac{By \cdot Cx - Bx \cdot Cy}{Ax \cdot By - Ay \cdot Bx}, \\ y_0 &= \frac{Ay \cdot Cx - Ax \cdot Cy}{Ay \cdot Bx - Ax \cdot By}, \\ R &= \sqrt{\frac{1}{W} \sum_{i=1}^{n} \left[(x_i - x_0)^2 + (y_i - y_0)^2 \right]}; \end{aligned} \tag{9}$$

kur:

$$\begin{aligned} Ay &= \sum_{i=1}^{n} w_i (y_i - \bar{y}) \cdot x_i , \\ By &= \sum_{i=1}^{n} w_i (y_i - \bar{y}) \cdot y_i , \\ Cy &= \frac{1}{2} \sum_{i=1}^{n} w_i (y_i - \bar{y}) \cdot (x_i^2 + y_i^2) , \\ Ax &= \sum_{i=1}^{n} w_i (x_i - \bar{x}) \cdot x_i , \end{aligned} \tag{10}$$

$$Bx = \sum_{i=1}^{n} w_i (x_i - \bar{x}) \cdot y_i ,$$

$$Cx = \frac{1}{2} \sum_{i=1}^{n} w_i (x_i - \bar{x}) \cdot (x_i^{\,2} + y_i^{\,2}) ,$$

kur:

$$\bar{x} = \frac{1}{W} \sum_{i=1}^{n} w_i x_i$$

$$\bar{y} = \frac{1}{W} \sum_{i=1}^{n} w_i y_i , \qquad (11)$$

$$W = \sum_{i=1}^{n} w_i .$$

Vienas iš svorių skaičiavimo metodų yra atvirkščiai proporcingas kvadratiniam nuokrypiui nuo kiekvieno taško (x_i , y_i) iki žingsniu prieš tai sutapatinto apskritimo:

$$w_i = \frac{1}{d_i^{\,2}}, \qquad (12)$$

kur : $$d_i = \sqrt{((x_i - x_0)^2 - (y_i - y_0)^2)} - R. \qquad (13)$$

Pradžioje nėra žinoma, kurie taškai priklauso apskritimui, todėl visi svoriai yra lygūs. Po pirmojo žingsnio visų taškų svoriai yra perskaičiuojami, priklausomai nuo nuokrypio iki išbrėžto apskritimo. Su naujai rastais taškų svoriais yra perskaičiuojami apskritimo parametrai x_0 , y_0, R.

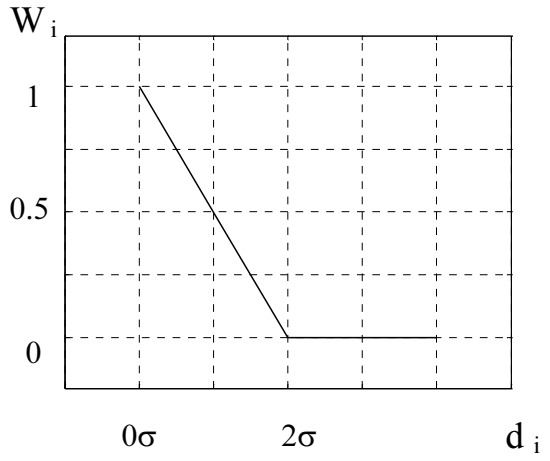
Tačiau yra geriau naudoti metodą, kuris visiškai neįvertina per daug nutolusių taškų. Remiantis tokiu metodu, svoris kiekvienam individualiam taškui yra randamas pagal:

$$\begin{cases} w_i = 1 - \dfrac{d_i}{2\sigma}, & \text{kai } \dfrac{1}{2} d_i < 2\sigma; \\ w_i = 0, & \text{kitais atvejais,} \end{cases} \qquad (14)$$

čia : $$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}. \qquad (15)$$

Tapdinimo ir naujų svorių skaičiavimo procedūra yra atliekama maksimaliai 8 kartus, arba sustabdoma tuomet, kai naujų rezultatų pokyčiai yra maži.

Grafiškai šis svorių skaičiavimo metodas pateiktas 3 paveiksle [2].



3 pav. Grafinis svorių skaičiavimo metodo vaizdas

1.4. Srautinių komandų skaičiavimo technologijos

Įprastai, procesoriai apdoroja tik vieną duomenų elementą su viena komanda. Procesoriai, turintys srautinių komandų įtaisą, pajėgia apdoroti daugiau nei vieną duomenį su viena komanda, pavyzdžiui: Intel Pentium 4 procesorius, kuris palaiko SSE2 srautinių komandų technologijas (žiūrėti priedas nr. 2) [7,3].

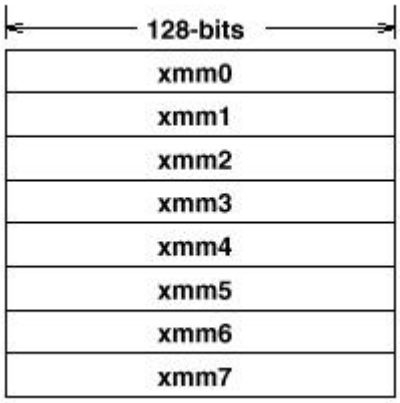
SSE2 registrai palaiko šiuos duomenų tipus (žiūrėti 1 lentelę) [5,9] :

- vieną pilna 128 bitų ilgio vertės.
- du - 64 bitų ilgio vertės;
- keturis - 32 bitų ilgio vertės;
- aštuonis - 16 bitų ilgio vertės;
- šešiolika - 8 bitų ilgio vertės;
- du dvigubo tikslumo slankaus kablelio skaičius, kurie yra 64-bitų ilgio vertės;
- keturis viengubo tikslumo slankaus kablelio skaičius, kurie yra 32-bitų ilgio vertės ;

1 lentelė. **SSE2 registruose palaikomi duomenų tipai [3]**

<table>
<tr><th>Duomenų tipai</th><th colspan="16">XMMx-registrai</th></tr>
<tr><td>1x 128 bitų vertė</td><td colspan="16">Bitai 127 .. 0</td></tr>
<tr><td>2x 64 bitų vertė</td><td colspan="8">64 bitai
registras 1</td><td colspan="8">64 bitai
registras 0</td></tr>
<tr><td>4x 32 bitų vertė</td><td colspan="4">32 bitai
registras 3</td><td colspan="4">32 bitai
registras 2</td><td colspan="4">32 bitai
registras 1</td><td colspan="4">32 bitai
registras 0</td></tr>
<tr><td>8x 16 bitų vertė</td><td colspan="2">16 bitai
registras 7</td><td colspan="2">16 bitai
registras 6</td><td colspan="2">16 bitai
registras 5</td><td colspan="2">16 bitai
registras 4</td><td colspan="2">16 bitai
registras 3</td><td colspan="2">16 bitai
registras 2</td><td colspan="2">16 bitai
registras 1</td><td colspan="2">16 bitai
registras 0</td></tr>
<tr><td>16x 8 bitų vertė</td><td>baitas 15</td><td>baitas 14</td><td>baitas 13</td><td>baitas 12</td><td>baitas 11</td><td>baitas 10</td><td>baitas 9</td><td>baitas 8</td><td>baitas 7</td><td>baitas 6</td><td>baitas 5</td><td>baitas 4</td><td>baitas 3</td><td>baitas 2</td><td>baitas 1</td><td>baitas 0</td></tr>
<tr><td>2x 64 bitų slankaus kablelio (floats) skaičiai</td><td colspan="8">64 bitų slankaus kablelio (floats) skaičiai
registras 1</td><td colspan="8">64 bitų slankaus kablelio (floats) skaičiai
registras 0</td></tr>
<tr><td>4x 32 bitų slankaus kablelio (floats) skaičiai</td><td colspan="4">32 bitų slankaus kablelio (floats) skaičiai
registras 3</td><td colspan="4">32 bitų slankaus kablelio (floats) skaičiai
registras 2</td><td colspan="4">32 bitų slankaus kablelio (floats) skaičiai
registras 1</td><td colspan="4">32 bitų slankaus kablelio (floats) skaičiai
registras 0</td></tr>
</table>

SSE2 turi 8 registrus, kurie vadinami xmm. Šių registrų išsidėstymas pavaizduotas 4 paveiksle.



4 pav. SSE2 registrai

SSE2 reikalauja operacinės sistemos palaikymo, kuri gali saugoti ir sugražinti procesoriaus reikiamą struktūrą. Šiuo metu, vienintelės operacinės sistemos, kurios palaiko SSE2 yra Microsoft Windows operacinės sistemos.

SSE2 komandos gali klasifikuotis į grupių komandas:

- perrašymo operacijos;
- duomenų kilnojimas;

- duomenų tvarkymas;
- aritmetinės operacijos;
- loginės operacijos;
- pastumimo operacijos.

Šios instrukcijos priemonės užtikrina veiksmų pagreitėjimą, ypač 3 – matėje grafikoje, fizikoje, realiame laike, video užšifravimui / iššifravimui, šifravimui ir moksliniuose tyrimuose [4,8].

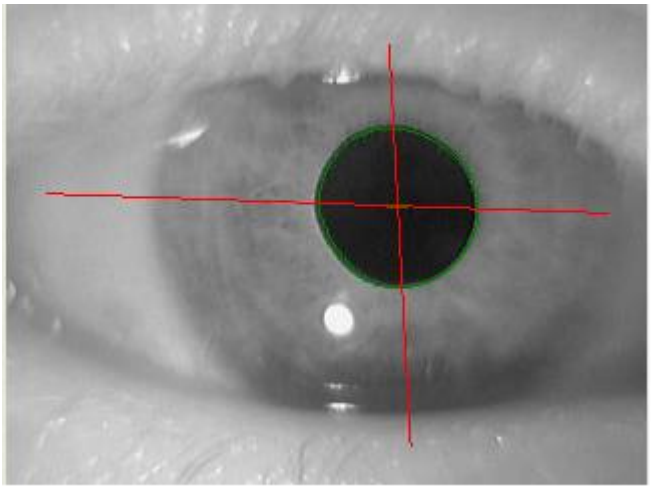
2. Programinė įrangos aprašymas

Aukščiau aptarti vyzdžio centro koordinačių nustatymo metodai bei jų skaičiavimo pagreitinimui panaudotos srautinės komandos yra įgyvendinti dviejose programinėse įrangose, kurios parašytos Borland C++ Builder 6 programavimo kalba. Šių programų veikimo principas analogiškas. Jos skiriasi tik tuo, kad vienoje iš jų yra panaudotos srautinės komandos (žiūrėti priedas nr. 1). Abi programinės įrangos buvo analizuojamos tuo pačiu kompiuteriu, kurio procesoriaus dažnis 3,2 Ghz.

Programai inicializavus, visų pirma yra įvedami video failai (kadrų sekos) ir atliekami pradinių sąlygų nustatymai, t.y. analizavimo metodo pasirinkimas (analizė tiesėmis ar analizė apskritimu). Nustačius pradines sąlygas, toliau yra vykdoma įvestų duomenų analizė. Po kiekvieno kadro analizės, gauti rezultatai yra atvaizduojami ekrane. Atlikus duomenų analizę, po kiekvieno kadro, yra gaunami rezultatai (kadro analizės laikas), kurie vėliau apdorojami.

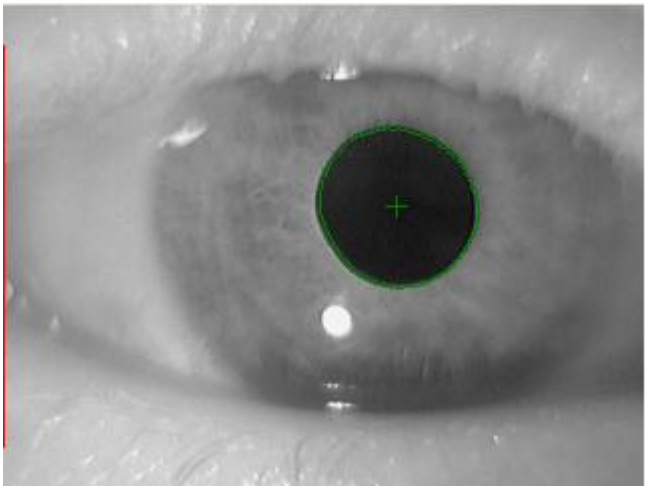
3. Tyrimo rezultatai

Išanalizavus kadrą analizės tiesėmis metodu, gauname 5 paveiksle parodytą akies vaizdą, kur dvi susikertančios raudonos tiesės pažymi akies vyzdžio centrą, o žali taškai – akies vyzdžio konturo taškus.



5 pav. Akies vyzdžio centro radimas analizės tiesėmis metodu

Išanalizavus kadrą analizės apskritimo metodu, gauname 6 paveiksle parodytą akies vaizdą, kur žalias kryžiupas žymi akies vyzdžio centrą.

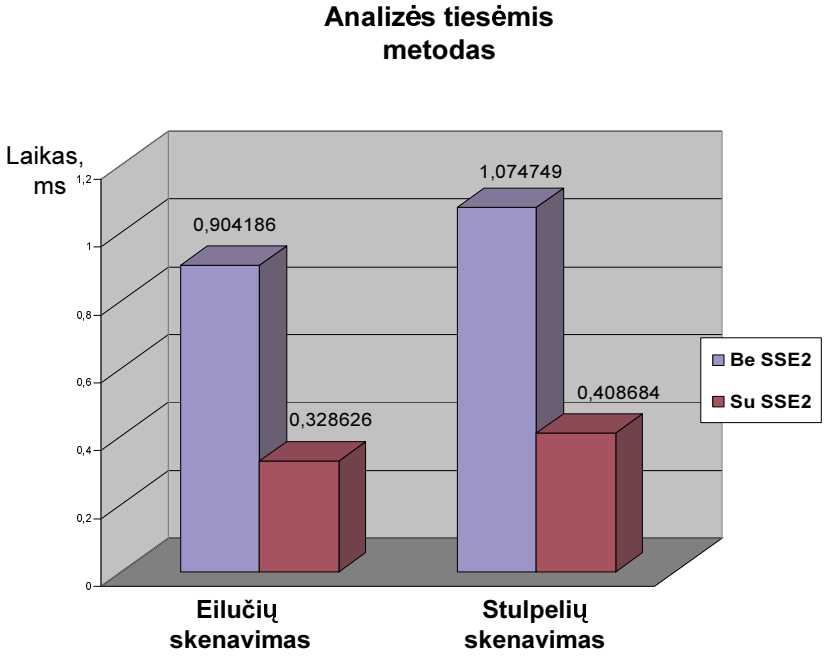


6 pav. Akies vyzdžio centro radimas analizės apskritimu metodu

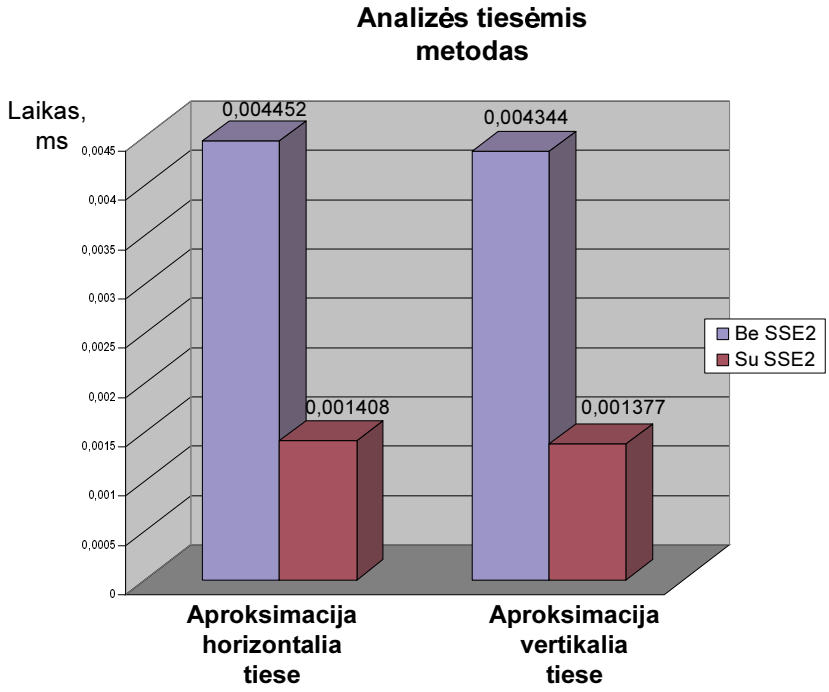
2 lentelė. **Akies vyzdžio centro koordinačių nustatymo metodų vidutiniai greičio įvertinimai**

	Analizės tiesėmis vidutinis kadro analizės laikas, ms				***Analizės apskritimu vidutinis kadro analizės laikas, ms***		
	Eilučių skenavimas	*Stulpelių skenavimas*	*Aproksimacija horizontalia tiese*	*Aproksimacija vertikalia tiese*	*Eilučių skenavimas*	*Stulpelių skenavimas*	*Aproksimacija apskritimu*
Be SSE2	*0,904186*	*1,074749*	*0,004452*	*0,004344*	*0,901592*	*1,071026*	*6,546695*
Su SSE2	*0,328626*	*0,408684*	*0,001408*	*0,001377*	*0,318045*	*0,439102*	*0,943017*

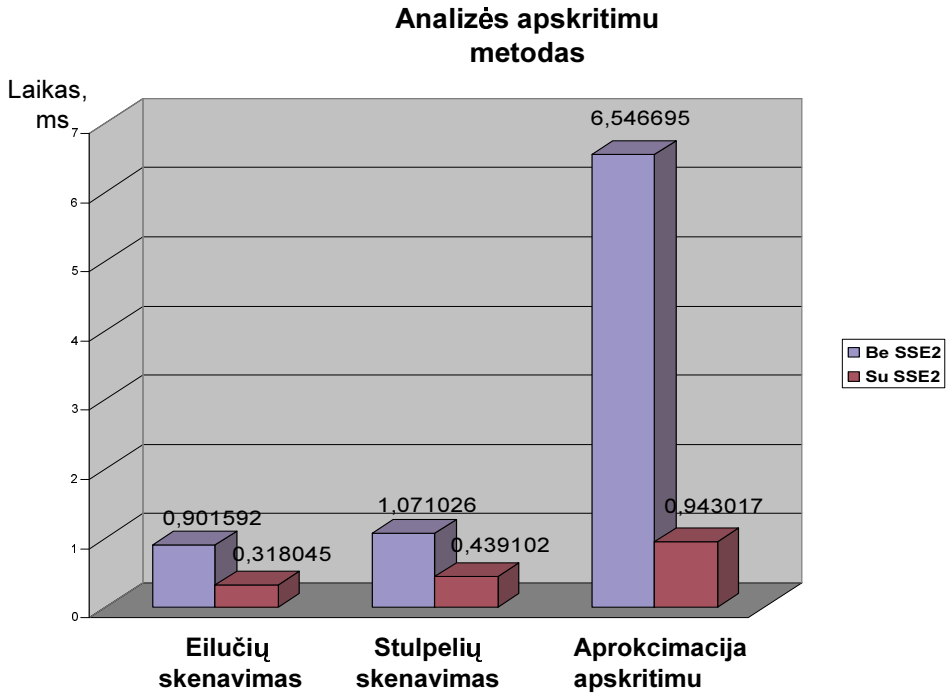
Lyginant 2 lentelės duomenis, matome, kad analizės tiesėmis metode, daugiausia laiko užtrunka eilučių ir stulpelių skenavimas, o analizės apskritimu metode daugiausia laiko užtrunka vyzdžio centro nustatymas, naudojantis aproksimaciją apskritimu.



7 pav. Analizės tiesėmis vidutinis kadro eilučių ir stulpelių skenavimo laikas



8 pav. Analizės tiesėmis vidutinis kadro aproksimacijos horizontalia ir vertikalia tiese laikas



9 pav. Analizės apskritimu vidutinis kadro eilučių ir stulpelių skenavimo laikas

3 lentelė. **Akies vyzdžio centro koordinačių nustatymo metodų laiko paklaidos dispersijos**

	Analizės tiesėmis laiko dispersija, ms				***Analizės apskritimu laiko dispersija, ms***		
	Eilučių skenavimas	*Stulpelių skenavimas*	*Aproksimacija horizontalia tiese*	*Aproksimacija vertikalia tiese*	*Eilučių skenavimas*	*Stulpelių skenavimas*	*Aproksimacija apskritimu*
Be SSE2	*0,001163*	*0,00000012*	*0,030978*	*0,00000032*	*0,0170129*	*0,0513794*	*0,0329436*
Su SSE2	*0,00029026*	*0,0000000088*	*0,00000602*	*0,000000019*	*0,00000354*	*0,0000017333*	*0,000238622*

4 lentelė. **Akies vyzdžio centro koordinačių nustatymo metodų laiko standartinės paklaidos įvertis**

	Analizės tiesėmis laiko sdartinės paklaidos įvertis, ms				***Analizės apskritimu laiko standartinės paklaidos įvertis, ms***		
	Eilučių skenavimas	*Stulpelių skenavimas*	*Aproksimacija horizontalia tiese*	*Aproksimacija vertikalia tiese*	*Eilučių skenavimas*	*Stulpelių skenavimas*	*Aproksimacija apskritimu*
Be SSE2	*0,034103*	*0,000346*	*0,176006*	*0,000566*	*0,130434*	*0,226670*	*0,181504*
Su SSE2	*0,017037*	*0,000094*	*0,002454*	*0,000138*	*0,001881*	*0,001317*	*0,015447*

Remiantis 3 ir 4 lentelės duomenimis, galime teigti, kad panaudojus srautines komandas akies vyzdžio centro koordinačių nustatymo metoduose, jų laiko greičio paklaidos dispersija bei laiko standartinės paklaidos įvertis žymiai sumažėja.

Išvados

Iš gautų rezultatų galima teigti, kad skaičiavimų pagreitėjimas yra žymus, panaudojus srautines komandas vyzdžio centro koordinačių nustatymo metoduose.

Kadrų eilučių ir stulpelių skenavimas tiek analizės tiesėmis , tiek analizės apskritimu metoduose vidutiniškai pagreitėja tris kartus. Naudojant analizės tiesėmis metodo aproksimacija horizontalia ir vertikalia tiese su srautinėmis komandomis laikas vidutiniškai pagreitėja keturis kartus, o analizės apskritimu metode atliekant aproksimaciją apskritimu laikas vidutiniškai pagreitėja šešis kartus.

Reziumė

D. Rimkus, Srautinių komandų taikymas videookulografijoje. Šiaulių universitetas, Technologijos fakultetas, Radiotechnikos katedra, 2005.

Šiame darbe videookulografiniams akių judesių matavimo metodams (analizė tiesėmis ir analizė apskritimu) buvo pritaikytos srautinės komandos (SSE2). Jų pagalba buvo siekiama paspartinti vyzdžio centro koordinačių metodų radimo laiką. Pateiktas vyzdžio centro aptikimo algoritmas, kuriame naudojami analizės tiesėmis ir analizės apskritimu metodai. Apžvelgtos srautinės komandos (SSE2) bei jų panaudojimo būdas. Atlikti tyrimai su gautais rezultatais, kurie parodė, kad srautinių komandų pagalba vyzdžio centro koordinačių nustatymo metodų laikas pagreitėjo.

Summary

D. Rimkus, The use of stream commands in videooculography. Šiauliai University, Faculty of Technology. Radioengineering departament. 2005.

There werw adapted the stream commands (SSE2) for videooculography measuring methods of eyes movements in this activity (an analysis by straight lines and an analysis by a circle). By their assistance the time of detection methods of eye pupil centre coordinates was accelerated. The algorithm of eye pupil centre detection was presented in which there were used the methods of analysis by straight lines and analysis by circles. There were presented stream commands (SSE2) and the methods of their use. After investigetion it was determined that the time of eye pupil centre coordinates by assistance of stream commands has been accelerated.

Literatūra

1. Chaudhuri B.B., Kundu P., Optimum circular fit to weighted data in multidimensional space // Pattern Recognition Letters. – 1993. – Vol.14. – P.1-6.
2. Ramanauskas N. Daunys G., The investigation of eye tracking accuracy using synthetic images // Electronics and Electrical Engineering. –2003. – Nr. 4 (46). P. 17–20.
3. Hayes Technologies, Technology: SIMD / MMX / SSE / SSE2 / 3Dnow! – 2002. Prieiga per internetą: http://www.hayestechnologies.com/en/techsimd.htm#SSE2
4. Intel Corporation, Using Streaming SIMD Extensions 2 (SSE2). – 2005. Prieiga per internetą: http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/19069.htm
5. Pabst T., Intel's New Pentium 4 Processor. – 2000. Prieiga per internetą: http://www4.tomshardware.com/cpu/20001120/p4-10.html#sse2_the_new_double_precision_streaming_simd_extensions
6. Kruusa I., Intel SIMD extensions. – 2004. Prieiga per internetą: http://www.tuleriit.ee/progs/index.php?rintel=1
7. Tommesani S., Intel SSE2. – 2000. Prieiga per internetą: http://www.tommesani.com/SSE2MMX.html
8. Intel Corporation, Using Streaming SIMD Extensions (SSE2) to Perform Big Multiplications. – 2000. Prieiga per internetą: http://cache-www.intel.com/cd/00/00/04/08/40809_w_big_mul.pdf
9. Microsoft Corporation, Streaming SIMD Extensions 2 Instructions. – 2005. Prieiga per internetą: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclang/html/vcrefwillamettenewinstructions.asp

Priedas 1

Eilučių skenavimas

```
void TForm1::Konturas_H(int flag)
{
BYTE *lpmatrica, *iB;
        float dfi1=0.;
        float dteta=0.;
        float x,  xp;
   int BR11, BR12, BR13, BR14;
   int delta1;
        int       iVyzdzioryskioriba=60;
   int i, isritis, j;
        int ix, iy;
        int j1, jp;
        int ntaskuk, ntasku1;
        int sp;
        float y[10];

lpmatrica=(LPBYTE)&bdata[0];

        ntasku=0;
        for  (i=0; i<f_height; i++)
   {
                isritis=0;
                for   (j=0; j<f_width; j++)

                {
                        iB=lpmatrica+i*f_width+j;
                        sp=*iB;
                        if ((sp<iVyzdzioryskioriba)&&(isritis==0))
                        {
                                BR11=j;
                                if (BR11>0) BR13=1; else BR13=0;
                                isritis=1;
                        }
                        else if (((sp>iVyzdzioryskioriba)||(j==f_width-1))&&(isritis==1)) //
j==169
                        {
                                BR12=j;
                                if (j==f_width-1) BR14=0; else BR14=1; //j==169
                                isritis=0;
                                delta1=BR12-BR11;
                                if ((delta1>=15)&&(delta1<=160)) // delta1 15 160
                                {
                                        ntasku++;
                                        BR1Y[ntasku]=i;
                                        BR1X[ntasku]=(float)BR11;
                                        IBR1[ntasku]=BR13;
                                        BR2Y[ntasku]=(float)i;
                                        BR2X[ntasku]=(float)BR12;
```

```
                                        IBR2[ntasku]=BR14;
                                    }     //if delta pabaiga
                                }         //else if pabaiga
                            }  // ciklo pagal j pabaiga
                        }    // ciklo pagal i pabaiga
}
```

Eilučių skenavimas, panaudojant srautines komandas

```
void TForm1::Konturas_H(int flag)
{
        BYTE *lpmatrica, *iB,*iB1,*iB2,*iB3,*iB4,*iB5,*iB6,*iB7,*iB8,*iB9,
     *iB10,*iB11,*iB12,*iB13,*iB14,*iB15,*iB16;
        float dfi1=0.;
        float dteta=0.;
        float x,  xp;
   int BR11, BR12, BR13, BR14;
   int delta1;
        //int       iVyzdzioryskioriba=60;
   int i, j;
        int ix, iy;
        int j1, jp;
        int ntaskuk, ntasku1;
        Byte sp,sp1,sp2,sp3,sp4,sp5,sp6,sp7,sp8,sp9,sp10,sp11,
      sp12,sp13,sp14,sp15,sp16,
       isritis,BR111,BR122, carry=0;
        float y[10];

I128_16 B[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
E[16]={255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255},
F[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16},
A[16]={59,59,59,59,59,59,59,59,59,59,59,59,59,59,59,59},
C[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
H[16]={60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60};

lpmatrica=(LPBYTE)&bdata[0];

        ntasku=0;


        for  (i=0; i<240; i++)
   {
                isritis=0x00;

           for   (j=0; j<304; j=j+16)
                {


                iB=lpmatrica+i*f_width+j;
                iB1=lpmatrica+i*f_width+j+1;
                iB2=lpmatrica+i*f_width+j+2;
                iB3=lpmatrica+i*f_width+j+3;
                iB4=lpmatrica+i*f_width+j+4;
                iB5=lpmatrica+i*f_width+j+5;
                iB6=lpmatrica+i*f_width+j+6;
                iB7=lpmatrica+i*f_width+j+7;
                iB8=lpmatrica+i*f_width+j+8;
                iB9=lpmatrica+i*f_width+j+9;
                iB10=lpmatrica+i*f_width+j+10;
```

```
                iB11=lpmatrica+i*f_width+j+11;
                iB12=lpmatrica+i*f_width+j+12;
                iB13=lpmatrica+i*f_width+j+13;
                iB14=lpmatrica+i*f_width+j+14;
                iB15=lpmatrica+i*f_width+j+15;

                B->i1=*iB;
                B->i2=*iB1;
                B->i3=*iB2;
                B->i4=*iB3;
                B->i5=*iB4;
                B->i6=*iB5;
                B->i7=*iB6;
                B->i8=*iB7;
                B->i9=*iB8;
                B->i10=*iB9;
                B->i11=*iB10;
                B->i12=*iB11;
                B->i13=*iB12;
                B->i14=*iB13;
                B->i15=*iB14;
                B->i16=*iB15;

asm
{
mov bl,isritis
cmp bl,0
jne LOOP2

MOVDQU xmm0,H
MOVDQU xmm1,B
pminub xmm1,xmm0
MOVDQU xmm0,A
//MOVDQU xmm1,B
pcmpgtb xmm1,xmm0
MOVDQU xmm0,E
andnpd xmm1,xmm0

MOVDQU XMM3,XMM1
MOVDQU XMM4,C
psadbw xmm3,xmm4
pextrw ebx,xmm3,0
pextrw eax,xmm3,4
or bl,al
cmp bl,0
JZ LOOP2

MOVDQU xmm0,F
andpd  xmm1,xmm0

//pextrw ebx,xmm1,7
//cmp bh,16
//JE LOOP4
```

```
LOOP1:
pextrw ebx,xmm1,0
cmp bl,0
psrldq xmm1,1
JZ LOOP1

//mov as[],bl
mov isritis,1
mov carry,1
mov BR111,bl
jmp LOOP4

//*****************************************************************
LOOP2:
mov bl,isritis
cmp bl,1
jne LOOP4

MOVDQU xmm0,H
MOVDQU xmm1,B
pminub xmm1,xmm0
MOVDQU xmm0,A
//MOVDQU xmm1,B
pcmpgtb xmm1,xmm0
//MOVDQU xmm0,E
//andnpd xmm1,xmm0

MOVDQU XMM3,XMM1
MOVDQU XMM4,C
psadbw xmm3,xmm4
pextrw ebx,xmm3,0
pextrw eax,xmm3,4
or bl,al
cmp bl,0
JZ LOOP4

MOVDQU xmm0,F
andpd  xmm1,xmm0

//pextrw ebx,xmm1,7
//cmp bh,16
//JE LOOP4

LOOP3:
pextrw ebx,xmm1,0
cmp bl,0
psrldq xmm1,1
JZ LOOP3

//mov as[],bl
mov isritis,0
mov carry,2
mov BR122,bl
LOOP4:
}
```

```
                    if ( isritis==1 && carry==1)
                    {
                    //isritis=0;
                    carry=0;
                    BR11=(int)BR111+j;

                    }else if (isritis==0 && carry==2)
                    {
                    BR12=(int)BR122+j;
                    carry=0;

                    j=320;

                            delta1=BR12-BR11;
                            if ((delta1>=15)&&(delta1<=160)) // delta1 15 160
                            {
                                    ntasku++;
                                    BR1Y[ntasku]=i;
                                    BR1X[ntasku]=(float)BR11;
                                    IBR1[ntasku]=1;
                                    BR2Y[ntasku]=(float)i;
                                    BR2X[ntasku]=(float)BR12;
                                    IBR2[ntasku]=1;


                            }
                        }

                }
        }
}
```

Stulpelių skenavimas

```
void TForm1::Konturas_V(int flag)
{
        BYTE *lpmatrica, *iB;
        float dfi1=0.;
        float dteta=0.;
        float x,  xp;
   int BR11, BR12, BR13, BR14;
   int delta1;
        int       iVyzdzioryskioriba=60;
   int i, isritis, j;
        int ix, iy;
        int j1, jp;
        int ntaskuk, ntasku1;
        int sp;
        float y[10];


        lpmatrica=(LPBYTE)&bdata[0];
        ntasku=0;
        for  (i=0; i<f_width; i++)
   {
                isritis=0;
                for   (j=0; j<f_height; j++)
                {
                        iB=lpmatrica+j*f_width+i;
                        sp=*iB;
                        if ((sp<iVyzdzioryskioriba)&&(isritis==0))
                        {
                                BR11=j;
                                if (BR11>0) BR13=1; else BR13=0;
                                isritis=1;
                        }
                        else if (((sp>iVyzdzioryskioriba)||(j==f_height-1))&&(isritis==1))
                        {
                                BR12=j;
                                if (j==f_height-1) BR14=0; else BR14=1;
                                isritis=0;
                                delta1=BR12-BR11;
                                if ((delta1>=20)&&(delta1<=120))
                                {
                                        ntasku++;
                                        BR1Y[ntasku]=(float)BR11;
                                        BR1X[ntasku]=(float)i;
                                        IBR1[ntasku]=BR13;
                                        BR2Y[ntasku]=(float)BR12;
                                        BR2X[ntasku]=(float)i;
                                        IBR2[ntasku]=BR14;
                                }     //if delta pabaiga
                        }       //else if pabaiga
                }  // ciklo pagal j pabaiga
```

```
        }   // ciklo pagal i pabaiga

}
```

Stulpelių skenavimas, panaudojant srautines komandas

```
void TForm1::Konturas_V(int flag)
{
        BYTE *lpmatrica, *iB,*iB1,*iB2,*iB3,*iB4,*iB5,*iB6,*iB7,*iB8,*iB9,
     *iB10,*iB11,*iB12,*iB13,*iB14,*iB15,*iB16;
        float dfi1=0.;
        float dteta=0.;
        float x,  xp;
   int BR11, BR12, BR13, BR14;
   int delta1;
        //int       iVyzdzioryskioriba=60;
   int i, j;
        int ix, iy;
        int j1, jp;
        int ntaskuk, ntasku1;
        Byte sp,sp1,sp2,sp3,sp4,sp5,sp6,sp7,sp8,sp9,sp10,sp11,
      sp12,sp13,sp14,sp15,sp16,
       isritis,BR111,BR122, carry=0;
        float y[10];




I128_16 B[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
E[16]={255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255},
F[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16},
A[16]={59,59,59,59,59,59,59,59,59,59,59,59,59,59,59,59},
C[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
H[16]={60,60,60,60,60,60,60,60,60,60,60,60,60,60,60,60};

        lpmatrica=(LPBYTE)&bdata[0];

        ntasku=0;


        for  (i=0; i<360; i++)
   {
                isritis=0x00;

           for   (j=0; j<224; j=j+16)
                {
                iB=lpmatrica+j*f_width+i;
                iB1=lpmatrica+(j+1)*f_width+i;
                iB2=lpmatrica+(j+2)*f_width+i;
                iB3=lpmatrica+(j+3)*f_width+i;
                iB4=lpmatrica+(j+4)*f_width+i;
                iB5=lpmatrica+(j+5)*f_width+i;
                iB6=lpmatrica+(j+6)*f_width+i;
                iB7=lpmatrica+(j+7)*f_width+i;
                iB8=lpmatrica+(j+8)*f_width+i;
                iB9=lpmatrica+(j+9)*f_width+i;
```

```
                iB10=lpmatrica+(j+10)*f_width+i;
                iB11=lpmatrica+(j+11)*f_width+i;
                iB12=lpmatrica+(j+12)*f_width+i;
                iB13=lpmatrica+(j+13)*f_width+i;
                iB14=lpmatrica+(j+14)*f_width+i;
                iB15=lpmatrica+(j+15)*f_width+i;

                B->i1=*iB;
                B->i2=*iB1;
                B->i3=*iB2;
                B->i4=*iB3;
                B->i5=*iB4;
                B->i6=*iB5;
                B->i7=*iB6;
                B->i8=*iB7;
                B->i9=*iB8;
                B->i10=*iB9;
                B->i11=*iB10;
                B->i12=*iB11;
                B->i13=*iB12;
                B->i14=*iB13;
                B->i15=*iB14;
                B->i16=*iB15;

asm
{
mov bl,isritis
cmp bl,0
jne LOOP2

MOVDQU xmm0,H
MOVDQU xmm1,B
pminub xmm1,xmm0
MOVDQU xmm0,A
//MOVDQU xmm1,B
pcmpgtb xmm1,xmm0
MOVDQU xmm0,E
andnpd xmm1,xmm0

MOVDQU XMM3,XMM1
MOVDQU XMM4,C
psadbw xmm3,xmm4
pextrw ebx,xmm3,0
pextrw eax,xmm3,4
or bl,al
cmp bl,0
JZ LOOP2

MOVDQU xmm0,F
andpd  xmm1,xmm0

//pextrw ebx,xmm1,7
//cmp bh,16
//JE LOOP4
```

```
LOOP1:
pextrw ebx,xmm1,0
cmp bl,0
psrldq xmm1,1
JZ LOOP1

//mov as[],bl
mov isritis,1
mov carry,1
mov BR111,bl
jmp LOOP4

//*****************************************************************
LOOP2:
mov bl,isritis
cmp bl,1
jne LOOP4

MOVDQU xmm0,H
MOVDQU xmm1,B
pminub xmm1,xmm0
MOVDQU xmm0,A
//MOVDQU xmm1,B
pcmpgtb xmm1,xmm0
//MOVDQU xmm0,E
//andnpd xmm1,xmm0

MOVDQU XMM3,XMM1
MOVDQU XMM4,C
psadbw xmm3,xmm4
pextrw ebx,xmm3,0
pextrw eax,xmm3,4
or bl,al
cmp bl,0
JZ LOOP4

MOVDQU xmm0,F
andpd  xmm1,xmm0

//pextrw ebx,xmm1,7
//cmp bh,16
//JE LOOP4

LOOP3:
pextrw ebx,xmm1,0
cmp bl,0
psrldq xmm1,1
JZ LOOP3

//mov as[],bl
mov isritis,0
mov carry,2
mov BR122,bl
LOOP4:
```

```
}
                        if ( isritis==1 && carry==1)
                        {
                        //isritis=0;
                        carry=0;
                        BR11=(int)BR111+j;
                        ListBox2->Items->Add((float)BR11);
                        //ListBox1->Items->Add((B->i2));
                        }else if (isritis==0 && carry==2)
                        {
                        BR12=(int)BR122+j;
                        carry=0;
                        //isritis=0x02;
                        j=240;
                        ListBox3->Items->Add((float)BR12);
                                delta1=BR12-BR11;
                                if ((delta1>=20)&&(delta1<=120)) // delta1 15 160
                                {
                                        ntasku++;
                                        BR1Y[ntasku]=BR11;
                                        BR1X[ntasku]=(float)i;
                                        IBR1[ntasku]=1;
                                        BR2Y[ntasku]=(float)BR12;
                                        BR2X[ntasku]=(float)i;
                                        IBR2[ntasku]=1;
                                }
                            }
                    }
                }
}
```

Aproksimacija horizontalia tiese

```
void TForm1::Aprok_tiese_H(float *X, float *Y)
{
        int j;
        h0=0;h1=0;
        float hh0, hh1;
        float SY=0, SX=0, SX2=0, SXY=0;
        for(j=1;j<=x_index;j++)
        {
                SY+=Y[j];
                SX+=X[j];
                SX2+=(X[j]*X[j]);
                SXY+=(Y[j]*X[j]);
        }
        hh0=(SY*SX2-SXY*SX)/(x_index*SX2-SX*SX);
        hh1=(x_index*SXY-SX*SY)/(x_index*SX2-SX*SX);

        if (hh1==0)
                hh1=0.00000000001;

        h0=-(hh0/hh1);
        h1=1/hh1;
}
```

Aproksimacija horizontalia tiese, panaudojant srautines komandas

```
void TForm1::Aprok_tiese_H(float *X, float *Y)
{

        int j;
        h0=0;h1=0;
        float hh0, hh1;
        float SY=0, SX=0, SX2=0, SXY=0;
      I128 A[4]={0,0,0,0,},B[4]={0,0,0,0,},C[4]={0,0,0,0},D[4]={0,0,0,0}
           ,E[4]={0,0,0,0},F[4]={0,0,0,0},G[4]={0,0,0,0},
           O1[4]={0,0,0,0},O2[4]={0,0,0,0};

    asm{xorpd xmm4,xmm4}
        for(j=1;j<=x_index;j++)
        {
                A->i1=X[j];
                A->i2=Y[j];
                B->i1=X[j];
                B->i2=X[j];
    asm{
         MOVDQU xmm2,A
         MOVDQU xmm3,B
         mulps xmm2,xmm3      // |0000|0000|  X[j]  | Y[j]  |
                               //  daugyba su
         MOVDQU B,xmm2        // |0000|0000| X[j]   | X[j]  |
                 }
         B->i3=Y[j];
         B->i4=X[j];
    asm{
         MOVDQU xmm2,B
         addps  xmm4,xmm2  //  susumuojame
         }                          // |Y[j]|X[j]|X[j]*X[j]|Y[j]*X[j]|


        }
     asm{MOVDQU C,xmm4} // isvedame gautus rezultatus
        SX2=C->i1;
        SXY=C->i2;
        SY=C->i3;
        SX=C->i4;

    A->i1=SY;A->i2=x_index;B->i1=SX2;B->i2=SXY;C->i1=SXY;
    C->i2=SX;D->i1=SX;D->i2=SY;E->i1=x_index;E->i2=x_index;
    F->i1=SX2;F->i2=SX2;G->i1=SX;G->i2=SX;
          asm
          {                             // hh0, hh1 skaiciavimas
          MOVDQU xmm2,A
          MOVDQU xmm3,B
          mulps  xmm2,xmm3
          MOVDQU xmm3,C
          MOVDQU xmm4,D
```

```
            mulps  xmm3,xmm4
            subps  xmm2,xmm3
            MOVDQU xmm4,E
            MOVDQU xmm5,F
            mulps  xmm4,xmm5
            MOVDQU xmm5,G
            mulps  xmm5,xmm5
            subps  xmm4,xmm5
            divps  xmm2,xmm4
            MOVDQU O1,xmm2
            }
            hh0=O1->i1;
            hh1=O1->i2;

        if (hh1==0)
                hh1=0.00000000001;

        h0=-(hh0/hh1);
        h1=1/hh1;
}
```

Aproksimacija vertikalia tiese

```
void TForm1::Aprok_tiese_V(float *X, float *Y)
{
        int j;
        v0=0;v1=0;
        float SY=0, SX=0, SX2=0, SXY=0;
        for(j=1;j<=y_index;j++)
        {
                SY+=Y[j];
                SX+=X[j];
                SX2+=(X[j]*X[j]);
                SXY+=(X[j]*Y[j]);
        }
        v0=(SY*SX2-SXY*SX)/(y_index*SX2-SX*SX);
        v1=(y_index*SXY-SX*SY)/(y_index*SX2-SX*SX);


}
```

Aproksimacija vertikalia tiese, panaudojant srautines komandas

```
void TForm1::Aprok_tiese_V(float *X, float *Y)
{
        int j;
        v0=0;v1=0;
        float SY=0, SX=0, SX2=0, SXY=0;
      I128 A[4]={0,0,0,0,},B[4]={0,0,0,0,},C[4]={0,0,0,0},D[4]={0,0,0,0}
          ,E[4]={0,0,0,0},F[4]={0,0,0,0},G[4]={0,0,0,0},
          O1[4]={0,0,0,0},O2[4]={0,0,0,0};

       asm{xorpd xmm4,xmm4}
        for(j=1;j<=y_index;j++)
        {

              A->i1=X[j];
              A->i2=Y[j];
              B->i1=X[j];
              B->i2=X[j];
     asm{
          MOVDQU xmm2,A
          MOVDQU xmm3,B
          mulps xmm2,xmm3     // |0000|0000|  X[j]  | Y[j]  |
                                 //  daugyba su
          MOVDQU B,xmm2       // |0000|0000| X[j]   | X[j]  |
                 }
          B->i3=Y[j];
          B->i4=X[j];
     asm{
          MOVDQU xmm2,B
          addps  xmm4,xmm2  //  susumuojame
          }                              // |Y[j]|X[j]|X[j]*X[j]|Y[j]*X[j]|

        }
      asm{MOVDQU C,xmm4} // isvedame gautus rezultatus
        SX2=C->i1;
        SXY=C->i2;
        SY=C->i3;
        SX=C->i4;

     A->i1=SY;A->i2=y_index;B->i1=SX2;B->i2=SXY;C->i1=SXY;
     C->i2=SX;D->i1=SX;D->i2=SY;E->i1=y_index;E->i2=y_index;
     F->i1=SX2;F->i2=SX2;G->i1=SX;G->i2=SX;
          asm
          {                              // v0, v1 skaiciavimas
          MOVDQU xmm2,A
          MOVDQU xmm3,B
          mulps  xmm2,xmm3
          MOVDQU xmm3,C
          MOVDQU xmm4,D
          mulps  xmm3,xmm4
```

```
            subps  xmm2,xmm3
            MOVDQU xmm4,E
            MOVDQU xmm5,F
            mulps  xmm4,xmm5
            MOVDQU xmm5,G
            mulps  xmm5,xmm5
            subps  xmm4,xmm5
            divps  xmm2,xmm4
            MOVDQU O1,xmm2
            }
            v0=O1->i1;
            v1=O1->i2;
}
```

Aproksimacija apskritimu

```
void TForm1::Aproks_apskritimu(int eye_flag) // aproksimacijos apskritimu algoritmas,
                                             // tikslinimas vyksta 8 kartus
{
        int X_korek, Y_korek;
        int i,j;
        float X0, Y0, R;
        float w[500],W;

        for (j=0; j<index; j++) {w[j]=1;}
                //W=0.1*index;

        for (i=1; i<=8; i++)
        {
                float d[500], d1[500], xmean=0, ymean=0, Ax=0, Bx=0, Cx=0, Ay=0, By=0,
Cy=0, dsdev=0, dmean=0;

                for (j=0; j<=index; j++)
                {
                        d[j]=0;
                        d1[j]=0;
                }

                for (j=0; j<index; j++)
                {
                        W+=w[j];
                        xmean+=x_konturo[j]*w[j];
                        ymean+=y_konturo[j]*w[j];
                }
                        xmean=xmean/W;
                        ymean=ymean/W;

                for (j=0; j<index; j++)
                {
                        Ax+=(x_konturo[j]-xmean)*w[j]*x_konturo[j];
                        Bx+=(x_konturo[j]-xmean)*w[j]*y_konturo[j];
                        Cx+=(x_konturo[j]-
xmean)*(pow(x_konturo[j],2)+pow(y_konturo[j],2))*w[j];

                        Ay+=(y_konturo[j]-ymean)*w[j]*x_konturo[j];
                        By+=(y_konturo[j]-ymean)*w[j]*y_konturo[j];
                        Cy+=(y_konturo[j]-
ymean)*(pow(x_konturo[j],2)+pow(y_konturo[j],2))*w[j];
                }
                        Cx=Cx/2;
                        Cy=Cy/2;

                X0=(By*Cx-Bx*Cy)/(Ax*By-Ay*Bx);
                Y0=(Ay*Cx-Ax*Cy)/(Ay*Bx-Ax*By);
```

```
for (j=0; j<index; j++)
{
        R+=w[j]*(pow(x_konturo[j]-X0,2)+pow((y_konturo[j]-Y0),2));
}

R=sqrt(R/(W+0.000001));


if (i==1)
{
        int deg;
        for (deg=0; deg<=360; deg++)
        {
                x1[deg]=X0+R*cos(deg*M_PI/180);
                y1[deg]=Y0+R*sin(deg*M_PI/180);
        }
}

if (i==8)
{
        int deg;
        for (deg=0; deg<=360; deg++)
        {
                x2[deg]=X0+R*cos(deg*M_PI/180);
                y2[deg]=Y0+R*sin(deg*M_PI/180);
        }
}




for (j=0; j<index; j++)
{
        d[j]=sqrt(fabs(pow(x_konturo[j]-X0,2)-pow(y_konturo[j]-Y0,2)))-R;
        d1[j]=d[j]/2;
}

// vektoriaus d, is index elementu,  standartines deviacijos skaiciavimas

  for (j=0; j<index; j++) dmean+=d[j];  // randamas vidurkis
dmean=dmean/index;

        for (j=0; j<index; j++)     // skaiciuojama deviacija dsdev
        dsdev+=pow(d[j]-dmean, 2);

        dsdev=sqrt(dsdev/(index-1));

        for (j=0; j<index; j++) // randami nauji svorio koeficientai kiekvienam taskui w[j]
        {
                if (d1[j]<2*dsdev)
                        w[j]=1- d[j]/2*dsdev;
                else
                        w[j]=0;
```

```
        }
        W=0;
}



if (eye_flag==0)
{
        XL[nkadro_nr]=X0;
        YL[nkadro_nr]=Y0;
        RL[nkadro_nr]=R;
}
else
{
        XR[nkadro_nr]=X0;
        YR[nkadro_nr]=Y0;
        RR[nkadro_nr]=R;
}
```

Aproksimacija apskritimu, panaudojant srautines komandas

```
void TForm1::Aproks_apskritimu(int eye_flag) // aproksimacijos apskritimu algoritmas,
                                             // tikslinimas vyksta 8 kartus
{
        int X_korek, Y_korek;
        int i,j;
        float X0, Y0, R;
        float w[500],W;




        for (j=0; j<index; j++) {w[j]=1;}
                //W=0.1*index;

        for (i=1; i<=8; i++)
        {
                float d[500], d1[500], xmean=0, ymean=0, Ax=0, Bx=0, Cx=0, Ay=0, By=0,
Cy=0,xx=0, yy=0, dsdev=0, dmean=0;
           I128 A[4]={0,0,0,0,},B[4]={0,0,0,0,},C[4]={0,0,0,0},D[4]={0,0,0,0}
           ,E[4]={0,0,0,0},F[4]={0,0,0,0},G[4]={0,0,0,0},J[4]={0,0,0,0},I[4]={0,0,0,0},
           O1[4]={0,0,0,0},O2[4]={0,0,0,0};

                for (j=0; j<=index; j++)
                {
                        d[j]=0;
                        d1[j]=0;
                }

           asm{
                xorpd xmm4,xmm4}

                for (j=0; j<index; j++)
                {
               A->i1=x_konturo[j];
               A->i2=y_konturo[j];
               B->i1=w[j];
               B->i2=w[j];

                asm{
                MOVDQU xmm2,A
                MOVDQU xmm3,B
                mulps xmm3,xmm2     // |0000|0000|  x_konturo[j]  | y_konturo[j]  |
                                    //  daugyba su
                MOVDQU B,xmm3       // |0000|0000|     w[j]      |     w[j]      |
                }
                B->i3=w[j];
                                    // |0000|w[j]|x_konturo[j]*w[j]|y_konturo[j]*w[j]|
                asm
                {
                MOVDQU xmm3,B
```

```
 addps  xmm4,xmm3  //  susumuojame
 }                  // |0000|w[j]|x_konturo[j]*w[j]|y_konturo[j]*w[j]|




 }
 asm{MOVDQU C,xmm4} // isvedame gautus rezultatus
 xmean=C->i1;
 ymean=C->i2;
 W=C->i3;
          xmean=xmean/W;
          ymean=ymean/W;

B->i1=xmean;
B->i2=xmean;
B->i3=ymean;
B->i4=ymean;

F->i1=xmean;
F->i2=ymean;
asm{
xorpd xmm3,xmm3
xorpd xmm7,xmm7}
  for (j=0; j<index; j++)
  {
A->i1=x_konturo[j];
A->i2=x_konturo[j];
A->i3=y_konturo[j];
A->i4=y_konturo[j];

C->i1=w[j];
C->i2=w[j];
C->i3=w[j];
C->i4=w[j];
D->i1=x_konturo[j];
D->i2=y_konturo[j];
D->i3=x_konturo[j];
D->i4=y_konturo[j];
E->i1=x_konturo[j];
E->i2=y_konturo[j];
G->i1=w[j];
G->i2=w[j];
J->i1=x_konturo[j];
J->i2=x_konturo[j];
I->i1=y_konturo[j];
I->i2=y_konturo[j];

asm
{                  //Ax, Bx, Ay, By skaiciavimas
MOVDQU xmm1,B
MOVDQU xmm2,A
```

```
subps  xmm2,xmm1   // |  x_konturo[j]  |  x_konturo[j]  |  y_konturo[j]  |  y_konturo[j]  |
                   //  skirtumas su
                   // |     xmean      |     xmean      |     ymean      |     ymean      |
MOVDQU xmm1,C
mulps  xmm2,xmm1   // |x_konturo[j]-xmean|x_konturo[j]-xmean|y_konturo[j]-ymean|y_konturo[j]-ymean|
                   //  dauginame su
                   // |    w[j]        |    w[j]        |    w[j]        |    w[j]     |
MOVDQU xmm1,D
mulps  xmm2,xmm1   // |(x_kon[j]-xmean)*w|(x_kon[j]-xmean)*w|(y_kon[j]-ymean)*w|(y_kon[j]-ymean)*w|
                   //  dauginame su
                   // |  x_konturo[j]   |  y_konturo[j]  |  x_konturo[j]  |    y_konturo[j] |
addps  xmm3,xmm2   //gautus rezultatus susumuojame

MOVDQU xmm4,E      // Cx, Cy skaiciavimas
MOVDQU xmm5,F
subps  xmm4,xmm5   // |0000|0000|  x_konturo[j]  |  y_konturo[j]  |
                   //  skirtumas su
                   // |0000|0000|     ymean      |     ymean      |
MOVDQU xmm5,G
mulps  xmm4,xmm5   // |0000|0000|x_konturo[j]-xmean|y_konturo[j]-ymean|
                   //  dauginame su
                   // |0000|0000|     w[j]       |      w[j]     |
MOVDQU xmm5,J
mulps  xmm5,xmm5   //  x_konturo[j] kelimas kvadratu
MOVDQU xmm6,I
mulps  xmm6,xmm6   //  y_konturo[j] kelimas kvadratu
addps  xmm5,xmm6
mulps  xmm4,xmm5
addps xmm7,xmm4    //gautus rezultatus susumuojame

}

asm{               // isvedame gautus duomenis
MOVDQU O1,xmm3
MOVDQU O2,xmm7}
Ax=O1->i1;
Bx=O1->i2;
Ay=O1->i3;
By=O1->i4;
Cx=O2->i1;
Cy=O2->i2;

 }
        Cx=Cx/2;
        Cy=Cy/2;

A->i1=By;A->i2=Ay;B->i1=Cx;B->i2=Cx;C->i1=Bx;
C->i2=Ax;D->i1=Cy;D->i2=Cy;E->i1=Ax;E->i2=Ay;
F->i1=By;F->i2=Bx;G->i1=Ay;G->i2=Ax;J->i1=Bx;
J->i2=By;
asm
{                  // X0, Y0 skaiciavimas
MOVDQU xmm2,A
MOVDQU xmm3,B
mulps  xmm3,xmm2
MOVDQU xmm2,C
```

```
MOVDQU xmm4,D
mulps  xmm4,xmm2
subps  xmm3,xmm4
MOVDQU xmm2,E
MOVDQU xmm4,F
mulps  xmm4,xmm2
MOVDQU xmm2,G
MOVDQU xmm5,J
mulps  xmm5,xmm2
subps  xmm4,xmm5
divps  xmm3,xmm4
MOVDQU O1,xmm3
}
X0=O1->i1;
Y0=O1->i2;



      for (j=0; j<index; j++)
      {
  A->i1=x_konturo[j];
  A->i2=y_konturo[j];
asm
   {
    MOVDQU xmm2,I
    subps  xmm2,xmm3
    mulps  xmm2,xmm2
    MOVDQU O2,xmm2
      }
   xx=O2->i1;
   yy=O2->i2;
              R+=w[j]*(xx+yy);
      }

      R=sqrt(R/(W+0.000001));


      if (i==1)
      {
              int deg;
              for (deg=0; deg<=360; deg++)
              {
                      x1[deg]=X0+R*cos(deg*M_PI/180);
                      y1[deg]=Y0+R*sin(deg*M_PI/180);
              }
      }

      if (i==8)
      {
              int deg;
              for (deg=0; deg<=360; deg++)
              {
```

```
                        x2[deg]=X0+R*cos(deg*M_PI/180);
                        y2[deg]=Y0+R*sin(deg*M_PI/180);
                }
        }



        for (j=0; j<index; j++)
        {
    A->i1=x_konturo[j];
    A->i2=y_konturo[j];
   asm
      {
       MOVDQU xmm2,I
       subps  xmm2,xmm3
       mulps  xmm2,xmm2
       MOVDQU O2,xmm2
      }
     xx=O2->i1;
     yy=O2->i2;
                d[j]=sqrt(fabs(xx-yy))-R;
                d1[j]=d[j]/2;
        }

        // vektoriaus d, is index elementu,  standartines deviacijos skaiciavimas

    for (j=0; j<index; j++) dmean+=d[j];  // randamas vidurkis
  dmean=dmean/index;

        for (j=0; j<index; j++)    // skaiciuojama deviacija dsdev
        dsdev+=pow(d[j]-dmean, 2);

        dsdev=sqrt(dsdev/(index-1));

        for (j=0; j<index; j++) // randami nauji svorio koeficientai kiekvienam taskui w[j]
        {
                if (d1[j]<2*dsdev)
                        w[j]=1- d[j]/2*dsdev;
                else
                        w[j]=0;
        }
        W=0;
}



if (eye_flag==0)
{
        XL[nkadro_nr]=X0;
        YL[nkadro_nr]=Y0;
        RL[nkadro_nr]=R;
}
```

```
        else
        {
                XR[nkadro_nr]=X0;
                YR[nkadro_nr]=Y0;
                RR[nkadro_nr]=R;
        }

}
```

Priedas 2

Srautinės komandos

Instruction class	Instruction	Operation
Data movement	movq	64 bit move from/to mmx register
	movd	32 bit move from/to mmx register
Data format conversion / movement	packsswb/dw	Parallel pack signed words into bytes / dwords into words with signed saturation
	packuswb/dw	Parallel pack unsigned words into bytes / dwords into words with unsigned saturation
	punpckhbw/wd/dq	Parallel unpack high 32 bits: bytes to words / words to dwords / dwords to qwords
	punpcklbw/wd/dq	Parallel unpack low 32 bits: bytes to words / words to dwords / dwords to qwords
Arithmetical operations	paddb/w/d	Parallel add for bytes / words / dwords
	psubb/w/d	Parallel subtraction for bytes / words / dwords
	paddsb/w	Parallel saturated add for signed bytes / words
	paddusb/w	Parallel saturated add for unsigned bytes / words
	psubsb/w	Parallel saturated subtraction for signed bytes / words
	psubusb/w	Parallel saturated subtraction for unsigned bytes / words
	pmaddwd	Parallel multiply signed words and add the results
	pmullhw	Parallel multiply signed words and store high 16 bits of results
	pmulllw	Parallel multiply signed words and store low 16 bits of results
	pcmpeqb/w/d	Parallel compare signed bytes / words / dwords for equality
	pcmpgtb/w/d	Parallel compare signed bytes / words / dwords for "greater than"
Logical operations	pand	Parallel and operation on all 64 bits
	pandn	Parallel and not operation on all 64 bits
	por	Parallel or operation on all 64 bits
	pxor	Parallel xor operation on all 64 bits
Shift operations	psllw/d/q	Parallel shift logical left words / dwords / qwords
	psraw/d	Parallel shift right signed words / dwords
	psrlw/d/q	Parallel shift right unsigned words / dwords / qwords
Enable FPU instructions	emms	Resets the FPU register states to enable FPU instructions

Prefetch and write-through instructions	prefetchx	Prefetch cache line into L1/L2/L3 cache
	movntq	Store data directly into main memory, bypassing the caches (MMX only)
	movntps	Store data directly into main memory, bypassing the caches (XMM only)
	maskmovq	Store individual bytes directly into main memory, bypassing the caches (MMX only)
XMM data movement	movaps	128 bit move from/to xmm register, memory data must be aligned
	movups	128 bit move from/to xmm register, memory data may be unaligned
	movss	32 bit move from/to xmm register
	movlps	64 bit move from/to lower 64 bit of xmm register
	movhps	64 bit move from/to high 64 bit of xmm register
	movlhps	64 bit move from lower 64 bit of xmm register to high 64 bit of xmm register
	movhlps	64 bit move from higher 64 bit of xmm register to lower 64 bit of xmm register
	movmskps	Extract upper bits of all dwords and store in a general register
Data format conversion / movement	pextrw	Extract word into general register
	pinsrw	Insert word from general register
	pmovmskb	Extract upper bits of all bytes and store in a general register
	pshufw	Shuffle words (MMX only)
	pshufps	Shuffle dwords (XMM only)
	unpckhps	Parallel unpack and interleave high dwords (XMM only)
	unpcklps	Parallel unpack and interleave low dwords (XMM only)
	cvtpi2ps	Parallel convert integer dwords to single precision floating point values (XMM only)
	cvtpi2ss	Convert integer dword to single precision floating point value (XMM only)
	cvtps2si	Parallel convert single precision floating point values to integer dwords (XMM only)
	cvtss2si	Convert single precision floating point value to integer dword (XMM only)
MMX arithmetical operations	pavgb/w	Parallel average for unsigned bytes / words
	pminub	Parallel minimum for unsigned bytes
	pmaxub	Parallel maximum for unsigned bytes
	pminsw	Parallel minimum for signed words
	pmaxsw	Parallel maximum for signed words

	pmulhuw	Parallel multiply unsigned words and store high 16 bits of results
	psadbw	Parallel absolute difference of unsigned bytes and sum up the results
XMM arithmetical operations	addps	Parallel add single precision floating point values
	addss	Add single precision floating point value in lower 32 bits
	subps	Parallel subtract single precision floating point values
	subss	Subtract single precision floating point value in lower 32 bits
	mulps	Parallel multiply single precision floating point values
	mulss	Multiply single precision floating point value in lower 32 bits
	divps	Parallel divide single precision floating point values
	divss	Divide single precision floating point value in lower 32 bits
	rcpps	Parallel approximate reciprocal single precision floating point values
	rcpss	Approximate reciprocal single precision floating point value in lower 32 bits
	sqrtps	Parallel square root single precision floating point values
	sqrtss	Square root single precision floating point value in lower 32 bits
	rsqrtps	Parallel approximate reciprocal square root single precision floating point values
	rsqrtss	Approximate reciprocal square root single precision floating point value in lower 32 bits
	minps	Parallel minimum of single precision floating point values
	minss	Minimum of precision floating point value in lower 32 bits
	maxps	Parallel maximum of single precision floating point values
	maxss	Maximum of precision floating point value in lower 32 bits
	cmpps	Parallel ompare single precision floating point values and return masks as result
	cmpss	Compare single precision floating point value in lower 32 bits and return mask as result
	comiss	Compare single precision floating point value in lower 32 bits and return result in flag register
	ucomiss	Compare unordered single precision floating point value in lower 32 bits and return result in flag register
XMM Logical operations	andps	Parallel and operation on all 128 bits
	andnps	Parallel and not operation on all 128 bits
	orps	Parallel or operation on all 128 bits
	xorps	Parallel xor operation on all 128 bits

Instruction class	Instruction	Operation
Prefetch and write-through instructions	movntdq / movntpd	Store data directly into main memory, bypassing the caches (XMM only)
	movnti	Store data in general register directly into main memory, bypassing the caches
	maskmovdqu	Store individual bytes directly into main memory, bypassing the caches (XMM only)
XMM data movement	movapd / movdqa	128 bit move from/to xmm register, memory data must be aligned
	movupd / movdqu	128 bit move from/to xmm register, memory data may be unaligned
	movsd	64 bit move from/to xmm register
	movlpd	64 bit move from/to lower 64 bit of xmm register
	movhpd	64 bit move from/to high 64 bit of xmm register
	movlhpd	64 bit move from lower 64 bit of xmm register to high 64 bit of xmm register
	movhlpd	64 bit move from higher 64 bit of xmm register to lower 64 bit of xmm register
	movmskpd	Extract upper bits of all qwords and store in a general register
Data format conversion / movement	pmovmskpd	Extract upper bits of all qwords and store in a general register
	pshuflw	Shuffle words in lower 64 bits (XMM only)
	pshufhw	Shuffle words in high 64 bits (XMM only)
	pshufpd	Shuffle dwords (XMM only)
	unpckhpd	Parallel unpack and interleave high qwords (XMM only)
	unpcklpd	Parallel unpack and interleave low qwords (XMM only)
	cvtpi2pd	Parallel convert integer dwords to double precision floating point values (XMM only)
	punpckhqdq	Parallel unpack high qwords
	punpcklqdq	Parallel unpack lower qwords
	cvtpi2sd	Convert integer dword to double precision floating point value (XMM only)
	cvtpd2si	Parallel convert double precision floating point values to integer qwords (XMM only)
	cvtsd2si	Convert double precision floating point value to integer qword (XMM only)
	cvtps2pd	Parallel convert single precision floating point values to double precision floating point values
	cvtpd2ps	Parallel convert double precision floating point values to single precision floating point values

	cvtss2sd	Convert single precision floating point value to double precision floating point value
	cvtsd2ss	Convert double precision floating point value to single precision floating point value
	cvtps2dq	Parallel convert single precision floating point values to signed dword integers
	cvtdq2ps	Parallel convert signed dword integers to single precision floating point values
	movq2dq	Move MMX register value to XMM register
	movdq2q	Move lower 64 bit of XMM register to MMX register
Integer arithmetical operations	paddq	Parallel add for 64 bit integers
	psubq	Parallel subtraction for 64 bit integers
	pmuludq	Parallel multiply 32 bit unsigned integers and return 64 bit result
XMM arithmetical operations	addpd	Parallel add double precision floating point values
	addsd	Add double precision floating point value in lower 64 bits
	subpd	Parallel subtract double precision floating point values
	subsd	Subtract double precision floating point value in lower 64 bits
	mulpd	Parallel multiply double precision floating point values
	mulsd	Multiply double precision floating point value in lower 64 bits
	divpd	Parallel divide double precision floating point values
	divsd	Divide double precision floating point value in lower 64 bits
	rcppd	Parallel approximate reciprocal double precision floating point values
	rcpsd	Approximate reciprocal double precision floating point value in lower 64 bits
	sqrtpd	Parallel square root double precision floating point values
	sqrtsd	Square root double precision floating point value in lower 64 bits
	rsqrtpd	Parallel approximate reciprocal square root double precision floating point values
	rsqrtsd	Approximate reciprocal square root double precision floating point value in lower 64 bits
	minpd	Parallel minimum of double precision floating point values
	minsd	Minimum of precision floating point value in lower 64 bits
	maxpd	Parallel maximum of double precision floating point values
	maxsd	Maximum of precision floating point value in lower 64 bits
	cmpps	Compare double precision floating point values and return masks as result

	cmpsd	Compare double precision floating point value in lower 64 bits and return mask as result
	comisd	Compare double precision floating point value in lower 64 bits and return result in flag register
	ucomisd	Compare unordered double precision floating point value in lower 64 bits and return result in flag register
XMM logical operations	andpd	Parallel and operation on all 128 bits
	andnpd	Parallel and not operation on all 128 bits
	orpd	Parallel or operation on all 128 bits
	xorpd	Parallel xor operation on all 128 bits
XMM shift operations	pslldq	Shift all 128 bits left
	psrldq	Shift all 128 bits rights, filling in zeros