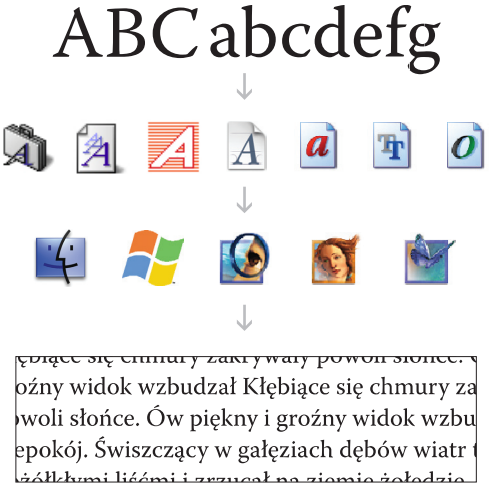
Font: cyfrowy nośnik pisma

ADAM TWARDOCH

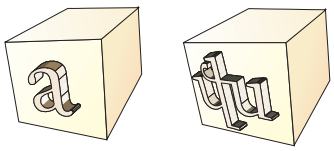
Co mają ze sobą wspólnego: praktycznie każdy zadrukowany w ciągu ostatnich kilku lat fragment papieru oraz w zasadzie każdy wyprodukowany w ostatnich latach monitor komputerowy? Między innymi to, że odwzorowane na nich litery pochodzą z fontów komputerowych. Fonty są ważnym elementem cyfrowego przetwarzania danych – stykają się z nimi na co dzień nie tylko osoby pracujące w branży DTP, lecz także zwykli użytkownicy komputerów. Niniejszy artykuł postara się przedstawić najważniejsze sprawy związane z użytkowaniem komputerowych fontów. Największą uwagę poświęcę przy tym formatom fontów PostScript Type 1, OpenType PS, TrueType oraz OpenType TT oraz zastosowaniu fontów w systemach operacyjnych Microsoft Windows 2000/XP oraz MacOS X. Jednak także użytkownicy starszych wersji systemów Windows i MacOS oraz użytkownicy innych platform powinni znaleźć w tym artykule odpowiedzi na najczęściej zadawane pytania związane z tematyką fontów komputerowych.



Rys. 1. Wybrane czynniki wpływajace na końcowy kształt graficzny tekstu: krój pisma, fonty, aplikacje.

Zacznijmy od tego, czym w ogóle jest font komputerowy. Ze względu na istniejące przeróżne formaty fontów oraz różny sposób stosowania ich w systemach operacyjnych i programach komputerowych, na drodze od kroju pisma do gotowego wizerunku konkretnego tekstu pojawić się mogą liczne komplikacje i pułapki – porównaj rys. 1.

Wyobraźmy sobie pudełko, w którym znajduje się kilkaset sześciennych klocków. Każdy klocek reprezentuje jeden znak pisma (tzw. glif). Na jednej ze ścian każdego klocka umieszczony jest wizerunek glifu. Jeżeli wyciągniemy z pudełka klocek taki, jak pokazany na rys. 2. po lewej stronie, to nie będziemy mieli problemu z odgadnięciem, że chodzi o literę *a*. Gdy jednak wyciągniemy klocek taki, jak pokazany po prawej stronie, to tylko nieliczni zgadną, że chodzi o ormiańską ligaturę *vn*.

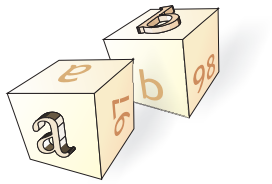


Rys. 2. Klocki z łacińską literą a oraz ormiańską ligaturą vn

Jeżeli naciskamy klawisz *a* na klawiaturze, to oczekujemy, że niezależnie od użytego kroju pisma na ekranie pojawi się mała litera *a*. Gdyby komputer za każdym razem musiał zgadywać, który z klocków w naszym pudełku zawiera wizerunek litery *a*, byłoby to bardzo nieefektywne. W związku z tym oczywiste staje się, że w naszym pudełku z klockami potrzebny jest jakiś porządek.

Znaki pisma w foncie uporządkowane są na dwa sposoby. Każdy znak pisma, oprócz określonej formy, posiada numer porządkowy oraz nazwę – zobacz rys. 3. Oczywiście samo ponumerowanie i ponazywanie znaków pisma w foncie nic by nie dało, gdyby nie istniały jakieś normy i zasady. Jeżeli w jednym foncie litera a nazywana byłaby *a* i nosiła numer 97, a w innym nosiłaby nazwę *letter_a* i posiadała numer 117, wówczas komputer i tak nie doszedłby z tym wszystkim do ładu.

Numer z literami, litery z numerem



Rys. 3. Znaki pisma w foncie są uporządkowane

Kwestia ponumerowania znaków pisma jest tak stara jak same komputery. Jako że urządzenia te wewnętrznie operują wyłącznie na liczbach, już projektanci pierwszych maszyn liczących musieli rozwiązać problem przechowywania tekstów w pamięci komputera. Z początku każde urządzenie miało swój własny, wewnętrzny schemat kodowania, według którego każdej literze alfabetu przyporządkowana była wartość numeryczna. Jednak dość szybko zauważono, że takie podejście do sprawy powoduje wiele komplikacji, gdy dane cyfrowe próbuje się przenosić między komputerami. W związku z tym ponad 35 lat temu amerykański komitet normalizacyjny ANSI podjął pracę nad opracowaniem standardu, pozwalającego na jednolite kodowanie znaków alfabetu i innych znaków pisarskich. Większość systemów komputerowych w tamtym czasie pracowała przy użyciu jednostek pamięci zwanych bajtami. Jeden bajt odpowiada ośmiu bitom i może przyjmować wartości całkowite w zakresie 0–255. Ze względów technicznych ANSI zdecydowało się na kodowanie siedmiobitowe i przypisało małe i wielkie litery alfabetu łacińskiego, cyfry i inne znaki pisarskie oraz zestaw specjalnych kodów sterujących liczbom z zakresu 0-127 – patrz rys. 4. Tak powstał kod ASCII (American Standard Code for Information Interchange), opublikowany w roku 1968 pod nazwą ANSI X3.4, a cztery lata później ogłoszony międzynarodowym standardem pod nazwą ISO 646.

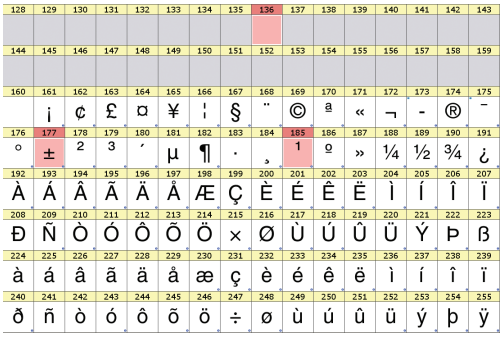
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Rys. 4. Zestaw znaków ASCII (ANSI X3.4, ISO 646), opublikowany w latach 1968–72.

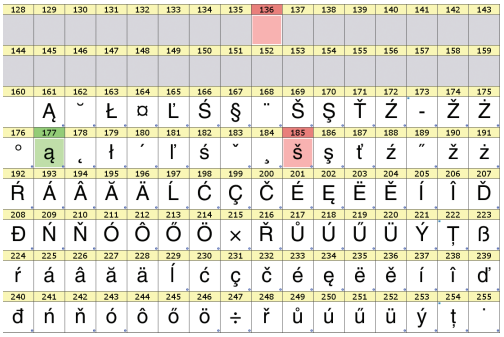
Jak nietrudno zauważyć, kod ASCII nadawał się tylko do kodowania bardzo prostych informacji tekstowych w języku angielskim. Ten zestaw znaków nie zawiera bowiem ani znaku myślnika (–), ani prawidłowych znaków cudzysłowu („polskich” bądź “angielskich”), ani znaków diakrytycznych – czyli używanych np. w języku francuskim, niemieckim, czeskim czy polskim liter wyposażonych w rozmaite akcenty, kropki, daszki czy ogonki – ani tym bardziej znaków alfabetów innych niż łaciński (np. cyrylicy lub greki).

Dynamiczny rozwój komputerów osobistych we wczesnych latach osiemdziesiątych sprawił, że wielu producentów komputerów i oprogramowania zauważyło ten mankament i zaczęło poszukiwać sposobu na zakodowanie znaków narodowych nie ujętych w ASCII. Jednym z pierwszych takich producentów był IBM, który swoje komputery osobiste wyposażył w zestaw znaków zwany codepage 437, w którym kodom 128–255 przypisano rozmaite znaki specjalne, głównie znaki diakrytyczne kilku języków zachodnioeuropejskich, znaki alfabetu greckiego oraz tzw. semigrafikę służącą do tworzenia ramek. Jakiś czas później, IBM wprowadził dodatkową stronę kodową 850, w której zamiast greki i części semigrafiki umieszczono dodatkowe znaki języków Europy Zachodniej. To był jednak dopiero czubek góry lodowej.

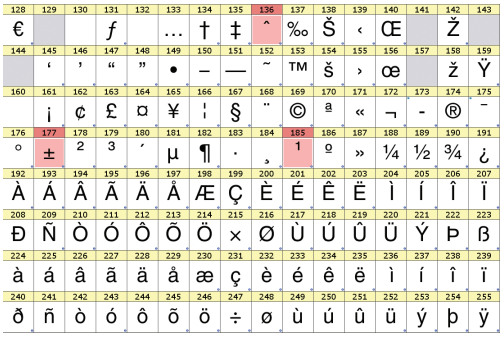
W połowie lat osiemdziesiątych europejskie zrzeszenie producentów sprzętu komputerowego ECMA opracowało pakiet stron kodowych, spośród których każda zawierała znaki określonego „obszaru językowego”: języków Europy Zachodniej, Wschodniej, Południowej i Północnej, oraz języków zapisywanych cyrylicą i greką, przy czym „dolna połowa” każdej ze stron kodowych odpowiadała kodowaniu ASCII. W 1987 roku organizacja ISO przy-



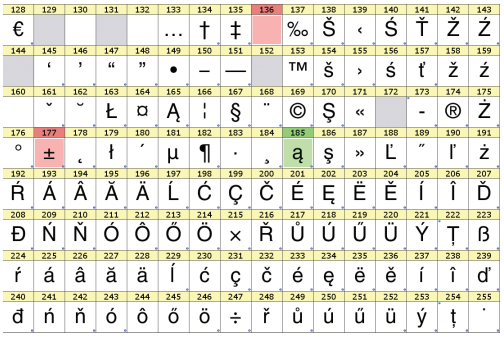
Rys. 5. Zestaw znaków ISO 8859-1 (ISO Latin 1), opublikowany w 1987 roku. Na kolejnych rysunkach kolorem czerwonym zaznaczono kody, którym przyporządkowano literę *ą* w innych stronach kodowych.



Rys. 6. Zestaw znaków ISO 8859-2 (ISO Latin 2), opublikowany w 1987 roku.



Rys. 7. Zestaw znaków MS Windows 1252 (Western), opracowany w latach 1990-92, a w roku w roku 1998 uzupełniony o znak waluty euro.



Rys. 8. Zestaw znaków MS Windows 1250 (Central European), opracowany w latach 1991-93, a w roku w roku 1998 uzupełniony o znak waluty euro.

jęła te strony kodowe jako międzynarodową normę pod nazwą ISO 8859. Podstawowy problem stron kodowych polegał na tym, że program komputerowy obrabiający tekst musiał „wiedzieć", w jakim kodowaniu zapisano tekst. Na rys. 5 i 6 przedstawiono „górne połowy" stron kodowych ISO 8859-1 i -2. W zależności od użytego kodowania, liczbie 177 przypisana jest litera ą lub znak plus-minus.

To jednak nie koniec problemów. Ze względu na powolność działania ISO oraz pewne wady norm 8859, w latach osiemdziesiątych producenci sprzętu i oprogramowania komputerowego masowo zaczęli opracowywać własne, niezgodne z trwającymi pracami normalizacyjnymi, strony kodowe. W Polsce, podobnie jak w innych krajach, producenci oraz sprzedawcy oprogramowania prześcigali się w opracowywaniu coraz to lepszych standardów kodowania umożliwiających umieszczanie znaków narodowych danego języka w miejsce wybranych znaków zachodnioeuropejskich – w ten sposób powstały strony kodowe Mazovia, DSH, Polonica i wiele innych. W większości przypadków te sposoby kodowania miały charakter lokalny i nie przetrwały próby czasu.

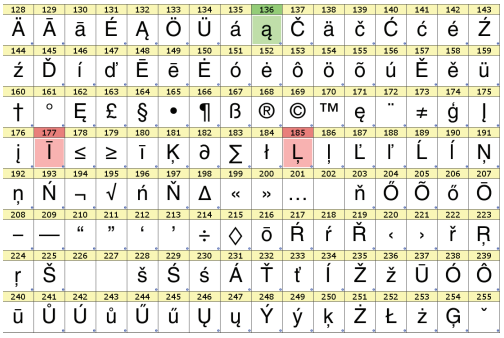
Wprowadzając na rynek system Windows, Microsoft wraz ze wspomnianym już komitetem ANSI opracował własny pakiet stron kodowych dla każdego obszaru językowego. Zachodnioeuropejska strona kodowa Microsoft (tzw. CP 1252) była rozszerzeniem kodowania ISO 8859-1, dodającym pewne znaki interpunkcyjne i znaki specjalne (w tym tak ważne cudzysłowy i myślniki, o których twórcy norm ISO „zapomnieli"). Jednak np. środkowo- czy też (jak ją wówczas nazywano) wschodnioeuropejska strona kodowa CP 1250 nie miała już z ISO 8859-2 nic wspólnego.

Firma Apple Computer, podobnie jak Microsoft, wprowadziła dla poszczególnych obszarów językowych własne strony kodowe. Kody znaków narodowych w systemie MacOS nie są zgodne ani z normami ISO, ani ze stronami kodowymi Microsoftu. Niestety, to jeszcze nie koniec problemów. Nie dość, że różne systemy kodowania przewidują różne kody dla różnych znaków narodowych w ramach danego obszaru językowego. Na dodatek owe obszary językowe różnią się między sobą.

Na przykład strona kodowa MacCE zawiera polską literę ą pod kodem 136, czeską literę r z haczykiem pod kodem 222 oraz łotewską literą g z przecinkiem pod kodem 174. Strona kodowa Windows 1250 zawiera polskie ą pod kodem 185 i czeskie r z haczykiem pod numerem 248, ale znaków łotewskich w tym kodowaniu brak – Microsoft umieścił je w bałtyckiej stronie kodowej 1258, która zawiera g z przecinkiem pod numerem 236, a nawet stosowane również w języku litewskim ą pod kodem 224, ale kodowanie to dla odmiany nie zawiera r z haczykiem. Oznaczało to nie mniej-ni więcej, że nie dało się przenieść wielojęzycznego tekstu napisanego po polsku, czesku i łotewsku z programu pracującego w MacOS do programu pracującego w Windows.

Nawet pracując w jednym tylko systemie operacyjnym i korzystając z jednego tylko programu, w przypadku składania tekstu wielojęzycznego możemy napotkać na duże trudności z kodowa-

Rys. 9. Zestaw znaków MacRoman, uzupełniony o znak waluty euro.



Rys. 10. Zestaw znaków MacCE – w tym kodowaniu miejsca dla znaku waluty euro nie przewidziano.

Pięć flakonów wody „Экземпляръ”.
tekst właściwy

Pięć flakonów wody „Ýęçĺěďë˙đú”.
tekst w kodowaniu Windows 1250 (Central European)

Рікж flakonyw wody „Экземпляръ”.
tekst w kodowaniu Windows 1251 (Cyrillic)

Rys. 11. Trudności z kodowaniem znaków w przypadku tekstów wielojęzycznych.

niem znaków. Skład wielojęzyczny z użyciem kodowania stronicowego (tj. z użyciem stron kodowych) wymaga stosowania wielu różnych fontów tego samego kroju, wśród których każdy zawiera określony podzbiór stosowanego zestawu znaków. Przygotowanie do druku tekstu wielojęzycznego np. przesłanego e-mailem lub przeniesionego z jakiegoś programu przez schowek wymaga od operatora zdwojonej uwagi – inaczej możemy napotkać na sytuację uwidocznioną na rys. 11.

Jednolicie

Problemy związane z kodowaniem znaków przy użyciu wielu różnych stron kodowych wskazały potrzebę stworzenia jednego, uniwersalnego zestawu znaków, posługującego się systemem kodowania opartym nie na pojedynczych bajtach, lecz na kodach używających dwu lub więcej bajtów.

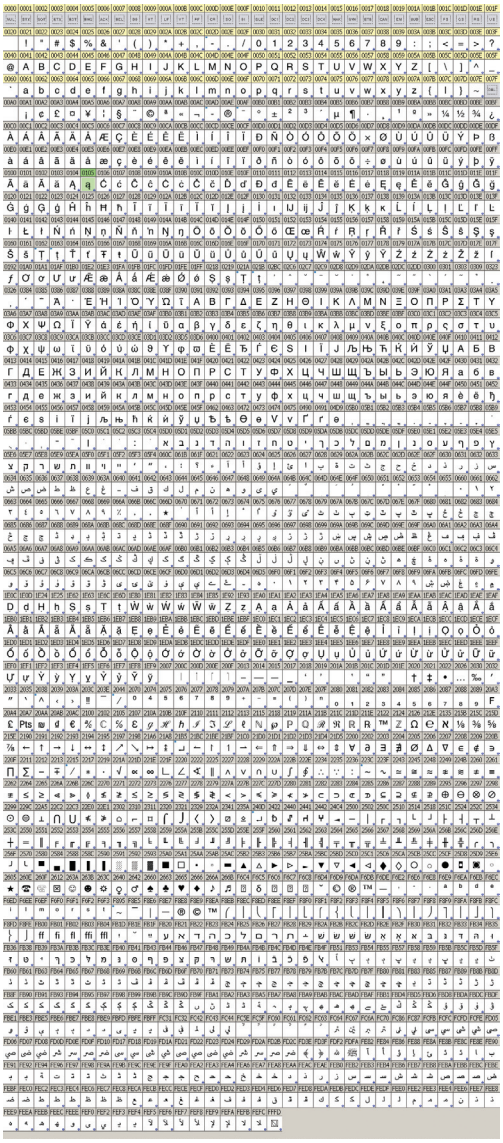
Już w roku 1984 grupa robocza ISO rozpoczęła prace nad opracowaniem uniwersalnego zestawu znaków. Trzy lata później kilka firm komputerowych, m.in. Xerox i Apple, założyło grupę roboczą Unicode, stawiającą sobie ten sam cel. W roku 1990 obydwie grupy opublikowały efekty swoich prac: DIS-1 10646 oraz Unicode 1.0. Ponieważ obie grupy uznały, że opracowywanie dwóch niezgodnych ze sobą „uniwersalnych” zestawów znaków nie ma sensu, rok później obie grupy nawiązały współpracę i stworzyły wspólny zestaw znaków, opublikowany pod nazwami Unicode 1.1 oraz ISO 10646. Od tej pory grupa robocza ISO oraz konsorcjum Unicode wspólnie pracują nad rozwojem uniwersalnego zestawu znaków oraz jednolitego sposobu ich kodowania pod nazwą Unicode.

Początkowo Unicode miał być standardem opartym na kodowaniu dwubajtowym, co dawało projektantom możliwość zagospodarowania 65 536 kodów. W roku 1996 opublikowana została wersja 2.0 standardu, zawierająca 38 885 znaków, w roku 1998 wydano wersję 2.1, dodając m.in. znak waluty euro. Wersja 3.0 standardu ujrzała światło dzienne w roku 1999, opisując repertuar 49 194 znaków.

Podczas prac nad kolejnymi wersjami konsorcjum Unicode doszło do wniosku, że limit 65 tys. znaków nie będzie wystarczający, w związku z czym podjęto decyzję, by Unicode uczynić standardem opartym na czterech bajtach, co – po uwzględnieniu pewnych ograniczeń technicznych – dało teoretyczną możliwość zakodowania ponad miliona znaków. Aby zapewnić zgodność z wczesnymi wersjami Unicode, standard podzielony został na 17 tzw. obszarów o numerach 0-16, po 65 536 znaków każdy. Zestaw znaków Unicode 3.0 umieszczony został w „domyślnym” obszarze 0, tzw. BMP (Basic Multilingual Plane).

W roku 2002 opublikowano wersję Unicode 3.2, opisującą 95 156 znaków, potem wersje 4.0 i 4.1, zaś obecnie trwają prace nad wersją 5.0 standardu. Wartości liczbowe kodów Unicode zapisywane są zwykle w systemie szesnastkowym, często poprzedzone prefiksem „U+” lub „0x” – na przykład litera ą ma w unikodzie numer U+0105 czyli w systemie dziesiętnym 261.

Repertuar znaków standardu Unicode stara się kodyfikować wszystkie znaki pisma używane w ogólnej komunikacji tekstowej.



Rys. 12. Fragment standardu kodowania Unicode.

Rys. 13. Znak pisma i różne glify z wizerunkiem tego znaku

Ogromną większość (ponad 90%) unikodu stanowią znaki ideograficzne języków azjatyckich: chińskiego, japońskiego i koreańskiego. Pozostałe kody obejmują znaki języków wszystkich kontynentów, posługujących się alfabetem łacińskim jak też innymi systemami pisma (alfabet cyrylicy, alfabet grecki, alfabet hebrajski, pismo arabskie, pisma hinduskie, sylabariusze inuickie itd.). Rysunek 12 ukazuje fragment repertuaru znaków Unicode złożony pismem Helvetica Linotype.

Znaki, nie glify

Znaki pisma nie posiadają ściśle określonego kształtu, a jedynie pewną strukturę, która określa układ i wzajemne relacje elementów. Ukazany na rysunku 13 znak pisma a może, w zależności od kroju i stylu pisma przyjmować różne kształty, zwane glifami. Standard Unicode stosuje rozróżnienie między pojęciami znak pisma (ang. character) oraz glif (ang. glyph). W tym rozumieniu znak pisma jest najmniejszą logiczną cząstką języka pisanego, podczas gdy glif jest odwzorowaniem konkretnego znaku pisma o określonych walorach graficznych i stylistycznych. Unikod opisuje kodowanie znaków pisma, nie zaś glifów – oznacza to na przykład, że glif z wizerunkiem małej litery a oraz glif z wizerunkiem kapitalika a kodowane są przy pomocy jednej i tej samej wartości: U+0061.

Elektroniczne czcionkarstwo

Przez stulecia litery pojawiały się w sposób czysto mechaniczny – były wynikiem odbicia na papierze wizerunku odlanego w specjalnym stemplu nazywanym czcionką. Dziś typowe czcionki należą już do przeszłości, zaś ich miejsce zajął nowy nośnik pisma – fonty. Fonty komputerowe to zbiory danych cyfrowych – obwiedni tworzących poszczególne znaki, zdefiniowanych przy użyciu równań matematycznych, a także licznych danych dodatkowych. Fonty takie, zwane często wektorowymi, mogą być swobodnie skalowane, dzięki czemu uzyskać możemy wizerunek znaku o dowolnych rozmiarach.

Proces tworzenia z fontu wizerunku litery zadanej wielkości nazywamy rastrowaniem, program, który za ten proces jest odpowiedzialny, nosi zaś nazwę rasteryzator (ang. rasterizer). Rasteryzatory są zwykle integralnymi elementami systemu zarządzania fontami (ang. type engine), do którego zadań należy m.in. łączenie odmian pism w rodziny, tworzenie w oparciu o dany font odmian pochodnych, dostęp do znaków specyficznych dla danego języka itp. Rastrowanie polega na wypełnianiu kropkami kształtu zdefiniowanego obwiednią. Kropki mogą być albo ekranowymi pikselami, albo kropelkami farby drukarskiej czy okruszkami tonera.

Na danym wycinku powierzchni „zadruku” można zmieścić ściśle określoną liczbę tychże kropek. Zależy to od tego, jakie materiały („farba” i podłoże, którym oczywiście może być luminofor ekranu monitora) użyte są w tym procesie, oraz od tego, jaką zastosowano technikę druku czy wyświetlania. Maksymalną liczbę kropek, jaką można przy użyciu konkretnego urządzenia (np. monitora, drukarki, naświetlarki, maszyny drukarskiej)

zmieścić na zadanym odcinku należącym do powierzchni „zadruku", nazywamy rozdzielczością tego urządzenia. Oczywiście im wyższa rozdzielczość urządzenia, tym lepiej, gdyż określonych rozmiarów obiekt można na nim przedstawić z uwzględnieniem większej liczby szczegółów. Przykładowo, rozdzielczość ekranu waha się w granicach 72-133 dpi (punktów na cal), podczas gdy papier zadrukowywany jest w rozdzielczościach od 600 dpi w przypadku drukarek do kilku tysięcy punktów na cal w przypadku profesjonalnych maszyn drukarskich.

Najpopularniejszymi, stosowanymi od lat już formatami fontów są: PostScript Type 1 oraz TrueType. Kilka lat temu pojawił się też nowy format, OpenType, dostępny w dwóch wariantach: OpenType PS oraz OpenType TT.

Niewiele osób wie, że światowa premiera pierwszego wektorowego formatu fontów odbyła się w Polsce. Na kongresie międzynarodowego stowarzyszenia typograficznego ATypI w Warszawie w roku 1975 dr Peter Karow, ówczesny dyrektor hamburskiej firmy URW, zaprezentował system Ikarus, służący do tworzenia skalowalnych fontów komputerowych. Do tego czasu fonty, stosowane głównie w maszynach fotoskładowych (komputerów osobistych jeszcze nie było) przechowywane były najczęściej w postaci rastrowej – każda wielkość pisma wymagała osobnej bitmapy. System Ikarus, składający się ze specjalizowanego komputera, ekranu oraz tabletu, a także specjalnego oprogramowania, umożliwiał operatorowi zamianę papierowych projektów krojów pism na wektorową postać cyfrową. Przy pomocy tego systemu przygotowane zostały pierwsze fonty wektorowe dla nowoczesnych systemów fotoskładu. Ale to dopiero rozwój komputerów osobistych i języka PostScript nadał impetu elektronicznemu czcionkarstwu.

Niecałą dekadę po pojawieniu się Ikarusa na rynku pojawiły się już pierwsze komputery Apple, w których po raz pierwszy zastosowane zostało przyjazne dla użytkownika graficzne środowisko, oparte na technologii wyświetlania obrazu zwanej QuickDraw. W tym samym czasie w sprzedaży znalazły się też i drukarki laserowe umożliwiające znakomitą jak na tamte czasy jakość druku w rozdzielczości 300 dpi. Firma Apple, zamierzająca wyprodukować własną drukarkę laserową przeznaczoną dla swoich komputerów, rozpoczęła poszukiwania możliwie uniwersalnego języka opisu strony, a więc odpowiednika QuickDraw, z tym, że przeznaczonego dla drukarek.

Próby stworzenia takiego języka podjęło kilka firm, spośród których Apple wybrał rewolucyjny jak na tamte czasy język PostScript, opracowany przez Adobe Systems, Inc., firmę założoną przez byłych pracowników koncernu Xerox. PostScript przynajmniej w teorii gwarantował możliwość niezależnego od typu urządzenia końcowego przygotowanie danych do druku. Do opisywania kształtów PostScript używał krzywych matematycznych trzeciego stopnia, tzw. krzywych Béziera.

Apple wyposażył w ten język swoją pierwszą drukarkę – Apple LaserWriter. Dzięki swym licznym zaletom, PostScript błyskawicznie zaczął stawać się bardzo popularny również w kręgach profesjonalnych, wypierając specyficzne rozwiązania poszczegól-

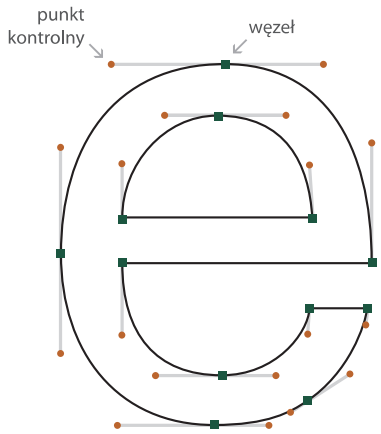
nych producentów. Jedną z nowości języka PostScript był format fontów Type 1, używający krzywych Béziera do opisu kształtu glifów i wyposażony w hinting – mechanizm polepszający wygląd liter w stosunkowo niskich rozdzielczościach drukarkowych (300 dpi). Gdy okazało się, że ogromna większość firm produkujących urządzenia drukarskie (drukarki i naświetlarki) zapragnęła wyposażyć swoje maszyny w PostScript, Adobe zaczęło domagać się wysokich opłat licencyjnych za interpreter PostScriptu, zawierający m.in. rasteryzator krojów Type 1. Specyfikacja formatu Type 1 trzymana była w tajemnicy, głównie ze względu właśnie na licencje, a także na opracowany przez Adobe hinting. Opublikowano tylko specyfikację formatu Type 3, który pozbawiony był tego cennego mechanizmu.

Wielcy producenci oprogramowania (Apple, IBM, Microsoft) zaczęli sobie jednak szybko zdawać sprawę, że warto by wykorzystać fonty skalowalne również w tworzonych przez nich systemach operacyjnych – do tej pory znaki na ekranie wyświetlane były w oparciu o fonty bitmapowe. Ponieważ specyfikacja formatu Type 1 nie była publicznie dostępna, nie chcąc uzależnić się od Adobe, w roku 1987 firmy Apple i Microsoft podjęły decyzję o opracowaniu własnego systemu wyświetlania grafiki i nowego formatu zapisu fontów, który między innymi w większym stopniu uwzględnić miał specyfikę wyświetlania znaków na ekranie.

W ramach tego porozumienia Microsoft rozpoczął prace nad opracowaniem podobnego zasadą działania do PostScriptu języka opisu strony o nazwie TrueImage, Apple miał zaś stworzyć nowy format zapisu fontów, któremu nadano tymczasową nazwę Royal. Prace Microsoftu nad TrueImage nie przyniosły oczekiwanych rezultatów, a więc wkrótce je zaprzestano – system był wolny, niedokładny, pełen błędów i nigdy nie doczekał się szerszej implementacji. Jednak informatycy Apple, na czele których stał fiński programista Sampo Kaasila, po dwóch latach prac stworzyli TrueType.

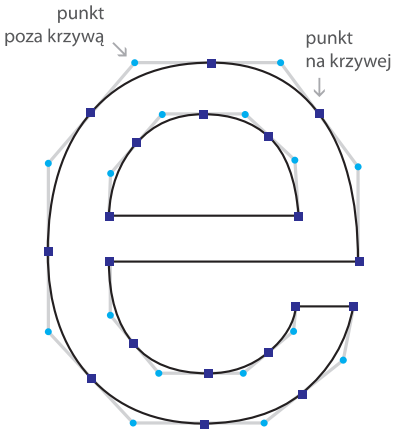
Aby ratować pozycję Type 1, która wydawała się być zagrożona nowo opracowywanym formatem, firma Adobe odtajniła w roku 1990 specyfikację tego formatu, i na początku 1991 roku stworzyła Adobe Type Manager (ATM), rasteryzator fontów Type 1 dla komputerów Apple. W tym samym okresie, w marcu 1991 roku, Apple zaprezentował format TrueType. Okazało się, że format ten dużo lepiej od Type 1 uwzględniał specyfikę wyświetlania znaków na ekranie – miał dużo lepszy mechanizm hintingu i znacznie szybszy rasteryzator. Firma Apple zakupiła więc od amerykańskiego koncernu poligraficznego Linotype licencje na wykorzystanie w swoim systemie operacyjnym krojów Times Roman, Helvetica i Courier, które już wcześniej znalazły się w repertuarze fontów dołączanych przez Adobe do interpretera PostScriptu. Inżynierowie firmy Apple stworzyli ich znakomite wersje TrueType, które reprezentacją na ekranie biły na głowę postscriptowe odpowiedniki.

Rok później technologia TrueType miała swoją drugą premierę – tym razem wchodząc w skład Windows 3.1 – nowej wersji systemu operacyjnego firmy Microsoft, która – we współpracy z drugim poligraficznym gigantem, Monotype – przygotowała



obrys PostScript
(krzywe 3. stopnia Béziera)

```
<path d="M 421 164 C 418 125 369 61 286
61 C 185 61 134 124 134 233 L 516 233 C
516 418 442 538 291 538 C 118 538 40 410
40 247 C 40 95 127 -15 274 -15 C 358 -15
392 5 416 21 C 482 63 506 140 509 164 Z M
134 303 C 134 384 199 459 279 459 C 386
459 420 384 425 303 Z" />
```



obrys TrueType
(krzywe 2. stopnia B-sklejane)

```
<path d="M 425 303 Q 418 389 380 423 Q
343 459 279 459 Q 215 459 175 414 Q 136
369 134 303 L 425 303 Z M 509 164 Q 494
83 429 30 Q 372 -16 279 -16 Q 169 -16 104
56 Q 40 126 40 247 Q 40 385 106 461 Q 173
538 291 538 Q 397 538 457 459 Q 516 379
516 233 L 134 233 Q 134 151 172 106 Q 210
61 286 61 Q 339 61 378 93 Q 417 126 421
164 L 509 164 Z" />
```

Rys. 14. Kształt glifu *e* kroju Helvetica, opisany przy użyciu obrysów PostScript i TrueType. U dołu tekstowa reprezentacja opisu kształtów wyrażona w języku SVG.

truetypowe wersje krojów Arial, Times New Roman i Courier New, zbliżonych do „oryginałów” zastosowanych w PostScripcie, jednak nieznacznie różniących się ze względów licencyjnych .

Niestety okazało się, że 16-bitowy rasteryzator Microsoftu (stworzony na bazie 32-bitowego programu Kaasili) miał sporo błędów. Na będących wówczas standardem komputerach z procesorami 286 bardziej skomplikowane znaki nie chciały się rastrować w dużych stopniach pisma. Również implementacja hintingu nie została dokonana poprawnie.

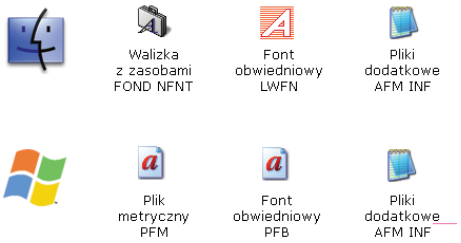
W związku z tym większość poważnych producentów pism drukarskich, na czele z Linotype, nie zdecydowała się na przeniesienie swoich krojów do formatu TrueType i zaczęła oferować je tylko w postaci fontów Type 1.

Glify w fontach postscriptowych zbudowane są z obwiedni opisanych krzywymi Béziera trzeciego stopnia. Każdy glif opisany jest w dwuwymiarowym układzie współrzędnych ograniczonym polem znaku (ang. em square). Krzywa Béziera definiowana jest przez uszeregowany ciąg węzłów (ang. nodes). Przebieg krzywych pomiędzy węzłami ustalany jest przy pomocy tzw. punktów kontrolnych (ang. control points). Każdy węzeł może mieć do dwóch punktów kontrolnych. Jako że każdy węzeł i punkt kontrolny opisany jest parami współrzędnych *x* i *y*, na każde dwa węzły może przypadać od 4 do 12 współrzędnych.

Kształt glifów w fontach truetypowych opisany jest obwiedniami zbudowanymi z krzywych B-sklejanych drugiego stopnia (ang. B-splines). Każda taka krzywa definiowana jest przez uszeregowany ciąg punktów, z których niektóre umieszczone są na krzywej (ang. on-curve points), a niektóre poza krzywą (ang. off-curve points). Matematyczny opis krzywych B-splines jest prostszy niż krzywych Béziera, więc glify fontów truetypowych mogą być kreślone szybciej niż glify postscriptowe. Ponieważ on-curve points i off-curve points najczęściej się przeplatają, na każde dwa punkty umieszczone na krzywej przypada od 4 do 8 współrzędnych. Ponieważ jednak punkty w krzywych B-splines muszą być ułożone gęściej niż węzły w krzywych Béziera, najczęściej opis glifu TrueType zajmuje nieco więcej miejsca niż opis glifu PostScript – zobacz rys. 14.

PostScript Type 1

Każdy font w formacie Type 1 składa się z dwóch części: obrysowej (obwiedniowej) i metrycznej. Część obrysowa zawiera opisane krzywymi Béziera kształty wszystkich glifów w foncie – jeden font Type 1 może zawierać nawet kilkadziesiąt tysięcy glifów. Najważniejszym wyróżnikiem każdego glifu jest jego nazwa. Nazwa glifu może się składać z małych i wielkich liter łacińskich, cyfr oraz kilku znaków specjalnych, i nie może przekraczać 32 znaków. Glify mogą mieć dowolne nazwy, ale najczęściej każdy glif będący wizerunkiem jakiegoś znaku pisma nosi nazwę zgodną ze standardem zalecanym przez Adobe. Dla przykładu, w myśl tego standardu, glif z wizerunkiem małej litery a nosi nazwę *a*, glif z wizerunkiem wielkiej litery B nazwę *B*, zaś glif z wizerunkiem wielkiej litery Ł nazwę *Lslash*. Część obrysowa fontu zawiera ponadto wizytówkę („metryczkę”) fontu zawierającą nazwę kro-



Rys. 15. Pliki, z których składa się font w formacie PostScript Type 1 w systemach MacOS i Windows.

ju pisma (w kilku wariantach) oraz informację o autorach (wpis Copyright). W tej części fontu zapisana jest też tabela kodująca, przyporządkowująca nazwy glifów kodom z zakresu 0-255.

Część metryczna fontu zawiera informacje o szerokościach pól znaków oraz listę par kerningowych (kerning to dosuwanie lub odsuwanie poszczególnych par znaków względem siebie dla poprawienia rytmu tekstu).

Pliki systemu MacOS

W większości systemów operacyjnych dane komputerowe zapisywane są na dysku w postaci ciągu bajtów zwanego plikiem. W systemach Unix, DOS czy Windows każdy plik jest litym ciągiem bajtów o określonej nazwie. Za analizę wewnętrznej struktury pliku odpowiedzialny jest program komputerowy przetwarzający plik. Informacja o rodzaju danych zapisanych w pliku przechowywana jest w postaci tzw. rozszerzenia nazwy pliku – kilkuliterowego (zwykle trzyliterowego) ciągu umieszczonego na końcu nazwy pliku, oddzielonego od niej kropką.

W systemie MacOS każdy plik dyskowy jest swoistym archiwum i składa się z dwóch niezależnych od siebie „teczek”: teczki zasobów (ang. resource fork) i teczki danych (ang. data fork). Część danych przechowywanych jest w teczce zasobów, część w teczce danych – aplikacja przetwarzająca plik decyduje o tym, w której teczce przechować informacje. Wadą takiego rozwiązania są uciążliwości przy wymianie plików z innymi systemami, zaletą możliwość łatwego dodawania do określonych porcji informacji różnego rodzaju metainformacji czytelnych dla systemu niezależnie od rodzaju pliku. Metainformacje te przechowywane są w teczce zasobów. Tu znajdują się m.in. dwa czteroliterowe ciągi (type i creator) opisujące rodzaj danych zapisanych w pliku, analogicznie do rozszerzenia nazwy w plikach windowsowych. Teczka zasobów może ponadto zawierać dowolną liczbę porcji informacji zwanych zasobami. Każdy taki zasób nosi czteroliterową nazwę i jest niejako „podplikiem” rezydującym wewnątrz makowego pliku dyskowego.

Fonty PostScript Type 1 występują w dwóch różnych wariantach, dostosowanych do pracy w systemach MacOS i Windows. W przypadku wariantu makowego, dane opisujące font znajdują się w teczkach zasobów kilku plików. Część obrysowa fontu znajduje się w zasobie POST wewnątrz pliku typu LWFN. Każda odmiana rodziny pism posiada osobny plik LWFN przechowujący część obrysową.

Części metryczne poszczególnych odmian rodziny pism znajdują się w zasobach FOND wewnątrz wspólnego dla rodziny pliku typu FFIL, zwanego – ze względu na systemową ikonkę – walizeczką. Zasoby FOND przechowują informację o kerningu, a także o kodowaniu i nazwach wszystkich odmian danej rodziny. Każdy zasób FOND posiada numer identyfikacyjny ID. System operacyjny MacOS wymaga, by numery FOND ID aktywnych w systemie fontów były unikatowe. Kiedy użytkownik próbuje instalować font o numerze FOND ID, który już jest zajęty, system automatycznie „przenumerowuje” zasób FOND – niestety czasami systemowi zdarza się przy tym uszkodzić plik fontu. Numer

```
<name>
  <namerecord nameID="0" platformID="1" platEncID="0">
    The digitally encoded machine readable software for
    producing the Typefaces licensed to you is copyrighted
    (c) 2001, 2002, Linotype Library GmbH or its
    affiliated Linotype-Hell companies. (...)
  </namerecord>
  <namerecord nameID="1" platformID="1" platEncID="0">
    Helvetica Linotype
  </namerecord>
  <namerecord nameID="2" platformID="1" platEncID="0">
    Regular
  </namerecord>
  <namerecord nameID="4" platformID="1" platEncID="0">
    Helvetica Linotype Regular
  </namerecord>
  <namerecord nameID="5" platformID="1" platEncID="0">
    Version 2.00   </namerecord>
<!-- ... -->
<cmap>
<cmap_format_4 platformID="3" platEncID="1" version="0">
  <map code="0x0" name=".null"/>
  <map code="0xd" name="nonmarkingreturn"/>
  <map code="0x20" name="space"/>
  <map code="0x21" name="exclam"/>
  <map code="0x22" name="quotedbl"/>
  <map code="0x23" name="numbersign"/>
<!-- ... -->
<glyf>
<TTGlyph name="e" xMin="82" yMin="-33" xMax="1057"
yMax="1102">
  <contour>
    <pt x="870" y="621" on="1"/>
    <pt x="856" y="795" on="0"/>
    <pt x="700" y="940" on="0"/>
    <pt x="571" y="940" on="1"/>
    <pt x="440" y="940" on="0"/>
    <pt x="279" y="756" on="0"/>
    <pt x="274" y="621" on="1"/>
  </contour>
```

```
%!PS-AdobeFont-1.0: HelveticaLTStd-Roman 003.002
%%CreationDate: Tue Apr 15 04:41:41 2003
%%VMusage: 120000 150000
11 dict begin
/FontInfo 14 dict dup begin
/version (003.002) readonly def
/Notice (Copyright © 1985, 1987, 1989, 1990, 1997,
1998, 1999, 2002 Adobe Systems Incorporated.  All
Rights Reserved. © 1981, 1999, 2002 Heidelberger
Druckmaschinen AG. All rights reserved.) readonly def
/FullName (HelveticaLTStd-Roman) readonly def
/FamilyName (Helvetica LT Std) readonly def
end readonly def
/FontName /HelveticaLTStd-Roman def
/Encoding 256 array
0 1 255 {1 index exch /.notdef put } for
dup 32 /space put
dup 33 /exclam put
dup 34 /quotedbl put
dup 35 /numbersign put
%% ...
/e {
40 556 hsbw
-15 76 hstem
233 70 hstem
459 79 hstem
0 94 vstem
385 91 vstem
381 164 rmoveto
-3 -39 -49 -64 -83 0 rrcurveto
-101 -51 63 109 hvcurveto
382 hlineto
185 -74 120 -151 vhcurveto
-173 -78 -129 -162 hvcurveto
-151 87 -111 147 vhcurveto
84 0 34 20 24 16 rrcurveto
66 44 24 74 3 25 rrcurveto
closepath
-375 139 rmoveto
81 64 75 81 vhcurveto
```

Rys. 16. Fragment kodu źródłowego fontu z krojem Helvetica w formacie TrueType (powyżej) i Type 1 (poniżej).

FOND ID jest też dla systemu źródłem informacji o zestawie znaków używanym w foncie.

Walizeczka może zawierać ponadto dodatkowe zasoby, np. bitmapowy font NFNT, używany przez starsze wersje systemu MacOS w określonych wielkościach ekranowych w miejsce zrastrowanego wizerunku pochodzącego z fontu obrysowego.

Wariant windowsowy fontu Type 1 przechowuje część obrysową w pliku binarnym o rozszerzeniu *.pfb*. Zawartość tego pliku jest identyczna z zawartością zasobu POST pliku LWFN stosowanego w fontach makowych. Część metryczna windowsowego fontu postscriptowego przechowywana jest w binarnym pliku *.pfm*. W odróżnieniu od makowej walizeczki, wspólnej dla całej rodziny fontów, w przypadku fontów windowsowych każda odmiana pisma posiada własny plik *.pfm*. Plik ten zawiera dane zbliżone do tych umieszczonych w zasobie FOND, a zatem kerning, szerokości pól znaków, nazwę kroju pisma oraz informację o zestawie znaków zawartym w foncie. Fonty Type 1 zawierają ponadto czasem dodatkowe pliki o rozszerzeniach *.afm* i *.inf*. Na ich podstawie program Adobe Type Manager może stworzyć brakującą część metryczną.

Fonty PostScript Type 1 mogą zawierać dowolną liczbę glifów, jednak dla większości aplikacji w najpopularniejszych systemach operacyjnych dostępnych jest ich maksymalnie 256.

Jest to znaczne ograniczenie, praktycznie uniemożliwiające obsługę wielu języków czy stosowanie ligatur lub form alternatywnych w jednym foncie. Jednym ze sposobów ominięcia tej bariery jest umieszczenie znaków danej odmiany w kilku fontach – tworzy się wówczas tzw. expert sets, zawierające kapitaliki, ligatury i formy alternatywne, albo fonty zakodowane według różnych stron kodowych, zawierające znaki różnych alfabetów. Istnieje co prawda rozszerzenie formatu Type 1 pod nazwą CID-keyed fonts, stworzone przez firmę Adobe na potrzeby rynków azjatyckich, które umożliwia stosowanie fontów zawierających kilka czy nawet kilkadziesiąt tysięcy znaków, jednak definicja tego formatu jest zawiła, a i tworzenie fontów CID jest skomplikowane.

Ciekawostką jest, że w formacie Type 3, który nie zawiera informacji o hintach, znaki mogą być kreślone przy użyciu pełnej palety instrukcji postscriptowych. Umożliwia to stosowanie rozmaitych „efektów specjalnych”. Np. w piśmie FF Beowolf firmy LettError wszystkie znaki podlegają w czasie drukowania ciągłej niewielkiej losowej modyfikacji, dzięki czemu ta sama litera za każdym razem przybiera minimalnie inny kształt.

TrueType

Format TrueType jest od formatu Type 1 dużo bardziej skomplikowany, posiada za to więcej możliwości i nietrudno go rozszerzać zachowując kompatybilność. Fonty TrueType zbudowane są w postaci serii tablic o czteroliterowych nazwach, z których każda opisuje inny aspekt fontu.

Fonty TrueType dostępne są, podobnie jak fonty Type 1, w dwóch wariantach. W przypadku wariantu makowego, dane dotyczące wszystkich odmian danej rodziny fontów zapisane są w teczce zasobów jednego pliku FFIL (walizeczce). Wewnątrz

teczki, każda odmiana fontu opisana jest przy pomocy zasobu sfnt. Dodatkowo, każdy font posiada również zasób FOND.

W przypadku wariantu windowsowego, każda odmiana pisma zapisana jest w postaci pliku o rozszerzeniu *.ttf*, przy czym zawartość tego pliku jest identyczna z zawartością zasobu sfnt fontów makowych.

Obwiednie glifów w fontach TrueType zapisane są w tablicy glyf przy pomocy krzywych B-sklejanych. Dzięki prostszej matematycznej formule fonty TrueType są szybciej kreślone na ekranie. Aby wydrukować taki font w urządzeniu postsciptowym, do niedawna system operacyjny musiał konwertować fonty TrueType do postaci Type 1, co nie zawsze odbywało się zgodnie z oczekiwaniami – czasem pojawiały się błędy, wynikające głównie z zaokrągleń przy konwersji współrzędnych, które oparte są na różnych skalach. Stąd utarło się przekonanie, że fonty TrueType nie nadają się do profesjonalnego przygotowywania publikacji, a tylko do zastosowań biurowych. Na opinię tę wpłynęła również fatalna jakość rasteryzatora TrueType dołączonego do Microsoft Windows 3.1.

Na szczęście praktycznie wszystkie nowe postscriptowe urządzenia drukarskie „rozumieją" także TrueType (wysyłane do drukarki w specjalnym formacie Type 42), zaś rasteryzator TrueType zawarty w Windows od wersji 95 jest znacznie poprawiony, a więc problem już nie istnieje.

Wraz z wprowadzeniem przez Microsoft w 1993 roku systemu Windows NT 3.1, fonty TrueType zaczęły obsługiwać standard kodowania Unicode. Zawarta w foncie tablica cmap przyporządkowuje poszczególnym glifom kody – przy czym nie istnieje tu znane z formatu Type 1 ograniczenie 256 wartości.

Dane metryczne i kerning fontów TrueType zapisane są w kilku osobnych tabelach, istnieje też sporo tabel dodatkowych, opisujących hinting, anti-aliasing i inne aspekty techniczne, a także zawierających liczne informacje na temat nazwy kroju pisma, projektanta, stanu autorskoprawnego i licencyjnego fontu itp.

Multiple Master

Poza formatem PostScript Type 1, firma Adobe opracowała bardzo ciekawy format Multiple Master (MM), umożliwiający dynamiczne tworzenie odmian danego kroju na drodze interpolowania wzdłuż osi między zadanymi odmianami krańcowymi. Osiami interpolacji mogą być m.in. grubość, szerokość, wielkość optyczna.

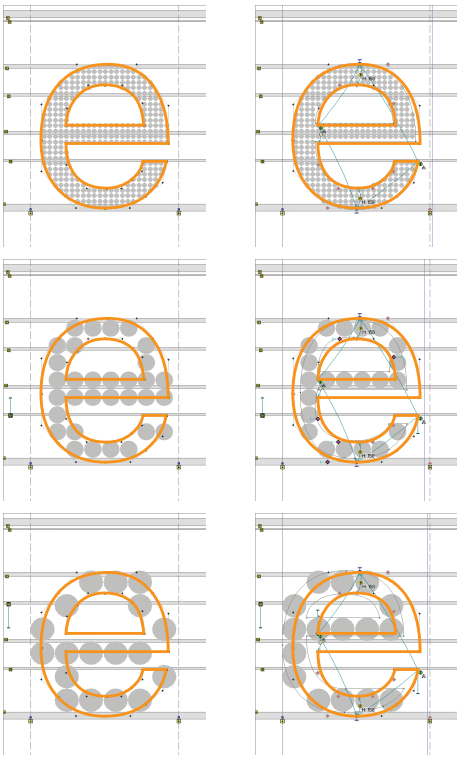
Faster **than light!**

Rys. 17. Przykłady zastosowania kroju Myriad MM (w formacie Multiple Master).

Przez wiele lat kroje w formacie Multiple Master tworzone były w zasadzie jedynie przez projektantów z firmy Adobe. Niestety w roku 1999 firma Adobe ogłosiła, że – ze względu na małe zainteresowanie rynku – wycofuje fonty Multiple Master ze produkcji, a docelowo również ze sprzedaży.

Ważny hinting

Jeżeli niewielkich rozmiarów obiekt, np. literę, wykreślimy na urządzeniu o niskiej rozdzielczości (np. ekran monitora), to może się okazać, że podczas rastrowania utracone zostaną liczne szczegóły.



Rys. 18. Porównanie rastrowania znaku *e* w foncie truetypowym Helvetica Linotype. Po lewej stronie font bez hintingu, po prawej z hintingiem ręcznym. Ukazany jest ten sam stopień pisma zrastrowany w różnych rozdzielczościach urządzenia wyjściowego (drukarka 300 dpi, ekran Windows 96 dpi, ekran Mac 72 dpi).

Typowy przykład: jeżeli jedna z nóżek litery *n* ma 149 jednostki szerokości, zaś druga 150, to po zrastrowaniu litery w rozdzielczości ekranu może się okazać, że w wyniku zaokrąglania jedna z nóżek ma 1 piksel grubości, zaś druga 2 piksele, co jest różnicą bardzo znaczącą. Jeszcze większe problemy pojawiają się przy rastrowaniu elementów ukośnych i łuków. Aby tych problemów uniknąć, producenci fontów stosują rozmaite techniki polepszające czytelność tekstów na ekranie. Do najpopularniejszych należą: hinting, wygładzanie krawędzi (ang. anti-aliasing), a także inne, np. ClearType.

Hinting w fontach Type 1 jest zbiorem „słów-wskazówek", które przypisywane są poszczególnym pionowym i poziomym elementom różnych liter. Rozmiar wszystkich elementów opatrzonych tym samym „słowem" powinien być w przypadkach wątpliwych zaokrąglany to tej samej liczby pikseli. Taki rodzaj hintowania nie pozwala projektantowi na dokładną kontrolę zachowania fontu na ekranie, a także znacznie spowalnia proces wyświetlania na ekranie. Hinting Type 1 nie rozwiązuje też problemów z rastrowaniem elementów ukośnych i okrągłych. Wymaga jednak od projektanta stosunkowo niewielkiego nakładu pracy.

W fontach TrueType można przy pomocy hintów kontrolować ze stuprocentową dokładnością każdy pojedynczy piksel pojawiający się na ekranie w każdej rozdzielczości. Innymi słowy, hinting TrueType jest wykonywany przez rasteryzator dokładnie tak, jak to zaplanował projektant fontu. Dzieje się tak za sprawą specjalnych instrukcji.

Projektant fontu ustala zależności pomiędzy poszczególnymi punktami kontrolnymi: może np. ustalić, że środkowa nóżka litery *m* powinna zawsze znajdować się dokładnie pośrodku nóżek skrajnych. Zestaw instrukcji hintujących TrueType jest jednak znacznie bogatszy niż w przypadku Type 1. Istnieją specjalne instrukcje hintujące elementy ukośne, przez co widocznie lepiej wyświetlane są truetypowe kursywy.

Poziom szczegółowy hintingu jest to zestaw programów definiowanych osobno dla każdego stopnia pisma w określonej rozdzielczości, mierzonej w pikselach na firet (ppm – pixel per em). Tzw. instrukcje delta pozwalają włączać i wyłączać poszczególne piksele. Jak widać na rysunku 18, im mniejsza rozdzielczość urządzenia (lub im mniejszy stopień pisma rastrowany na danym urządzeniu), tym hinting zaczyna odgrywać ważniejszą rolę. Dolny przykład rastrowania pokazuje, że font bez hintingu staje się w tej rozdzielczości praktycznie nieczytelny.

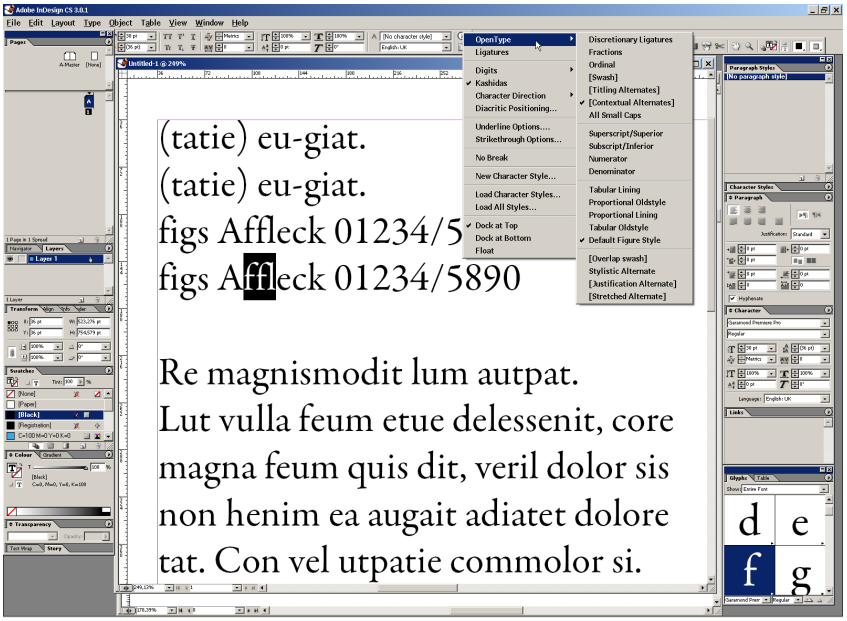
Hintowanie TrueType wymaga znacznie większego nakładu pracy ze strony projektanta – za to efekty są znacznie lepsze niż w przypadku hintów Type 1. Technologia hintingu stosowana w fontach TrueType ma też tę zaletę, że nie wymaga od rasteryzatora żadnych dodatkowych obliczeń, co znacznie przyspiesza proces rastrowania.

Oznacza to, że tak doskonałą czytelność liter na ekranie w małych stopniach pisma, jaką osiągnięto w przypadku fontów firmy Microsoft (Verdana, Georgia, Trebuchet), osiągnąć można tylko stosując TrueType.

Type 1 czy TrueType?

Niewątpliwą przewagą formatu TrueType nad Type 1 jest uniwersalność tego pierwszego – nadaje się on zarówno do definicji fontów „dużych” (zawierających znaki języków azjatyckich), jak i „małych” (zawierających litery alfabetów europejskich). Format TrueType jest także w pełni skalowalny (rozszerzalny), co oznacza, że nowo opracowane fonty, zawierające np. znaki Unicode, działają poprawnie również na starych urządzeniach i systemach (w zakresie jaki dopuszcza „stary” rasteryzator). Właśnie z tego względu na strukturze plików TrueType oparto nowy format OpenType.

Format Type 1 ma jednak również sporo niepodważalnych zalet. Jego budowa jest przede wszystkim prosta i przejrzysta, co umożliwia tworzenie dużo mniejszym nakładem środków zarówno fontów w tym formacie, jak i narzędzi do obróbki tychże. Dzięki swej prostocie, łatwiejsza staje się kontrola wewnętrznej struktury fontu Type 1, łatwiej też można usunąć ewentualne błędy. Format krzywych, z których konstruowane są obwiednie znaków, zgodny jest z ogólnie przyjętym standardem, stosowanym w języku PostScript i w wielu programach ilustracyjnych.



Rys. 19. Dostęp do funkcji zecerskich OpenType w programie Adobe InDesign.

Co do hintingu, to twórca fontów Type 1 może szybko osiągnąć przyzwoity efekt, ale nic ponadto, z drugiej zaś strony przygotowując font TrueType można stworzyć dzieło wprost znakomite, jednak nieproporcjonalnie większym nakładem pracy.

OpenType

W roku 1999 firmy Microsoft i Adobe wprowadziły na rynek nowy format fontów: OpenType, będący połączeniem fontów TrueType i PostScript Type 1. Fonty OpenType dostępne są w dwóch wariantach: OpenType TT (w których kształty znaków opisane są krzywymi B-spline, rozszerzenie nazwy pliku *.ttf*) oraz OpenType PS (w których glify opisane są krzywymi Béziera, rozszerzenie nazwy pliku *.otf*).

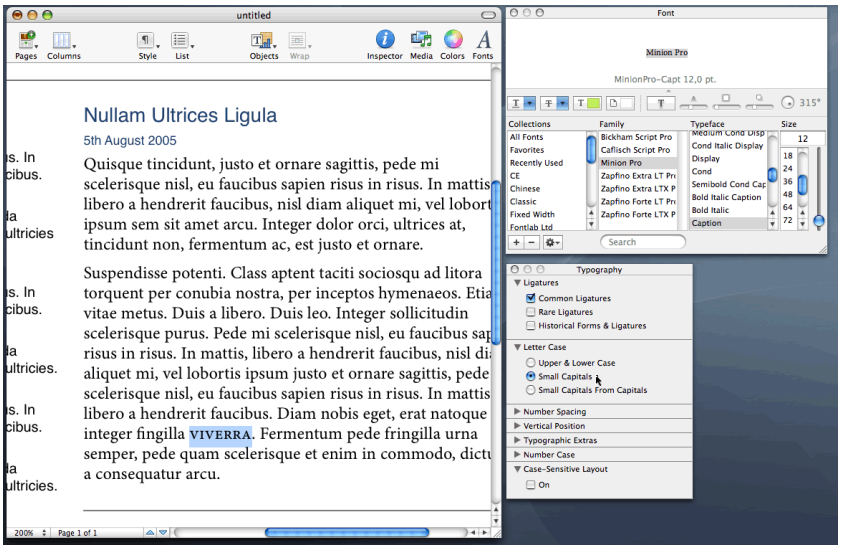
Fonty w formacie OpenType PS mogą być stosowane w praktycznie wszystkich obecnie stosowanych systemach operacyjnych: w Mac OS X oraz Microsoft Windows 2000 i XP bez instalacji jakichkolwiek dodatków, a także w Mac OS 8, 9 i Classic po instalacji programu Adobe Type Manager (ATM) Light 4.6 oraz w Microsoft Windows 95, 98, ME i NT 4.0 po instalacji programu ATM Light 4.1. Fonty w formacie OpenType TT obsługiwane są bez jakichkolwiek dodatków we wszystkich systemach Microsoft Windows oraz w Mac OS X – ale nie w Mac OS 9.

Nowatorską cechą fontów OpenType są tzw. funkcje zecerskie (ang. OpenType Layout features).

Funkcje zecerskie OpenType wymyślone zostały, aby w ogóle umożliwić pisanie tekstu w niektórych językach zgodnie z ich ortografią. Języki oparte na systemach pisma takich jak arabski, hebrajski, dewanagari czy tajski (i wiele innych) mają zasady ortograficzne bardziej skomplikowane od językach używających alfabetu łacińskiego. Wiele z tych systemów pisma to systemy pisma łączonego, gdzie kształt liter zmienia się w zależności od położenia.

W alfabecie łacińskim istnieją dwie podstawowe formy liter: wielkie i małe. W piśmie arabskim dla każdej litery istnieją cztery takie formy – początkowa, środkowa, końcowa i wolno stojąca. Rysunek 21 ukazuje u góry słowo arabskie złożone litera po literze ($l + h + q$) z form wolno stojących. Oczywiście górna czerwona forma jest niepoprawna. Na tym samym rysunku u dołu ukazane jest to samo słowo, ale złożone z użyciem form odpowiednio dobranych w zależności od kontekstu (początkowe *l*, środkowe *h* i końcowe *q*). Dolna czerwona forma pokazuje, jak prawidłowo powinno wyglądać to słowo.

W piśmie łacińskim wielkie i małe litery niosą ze sobą różnicującą informację sematyczną („kowalski” i „Kowalski” to co innego). W związku z tym naturalne jest, że litery te kodowane są osobno. W piśmie arabskim jest inaczej – nie istnieje „kasztowość” pisma czyli rozróżnienie liter wielkich i małych. Litera *h* pozostaje literą *h* niezależnie od tego, czy czy znajduje się na początku, w środku czy na końcu wyrazu. Jedynie forma zewnętrzna tego znaku się zmienia. Dlatego każda litera arabska zakodowana jest w tekście przy użyciu jednego kodu (Unicode). Natomiast zadaniem procesora tekstu jest odpowiednia poprawna prezentacja tekstu na ekranie. Oczywisty przykład: jeżeli wstawię kursor w środek arabskiego słowa i nacisnę spację, wówczas otaczające kursor litery powinny automatycznie się zmienić z form środkowych w, odpowiednio, początkową i końcową, a między nimi powinna pojawić się spacja.

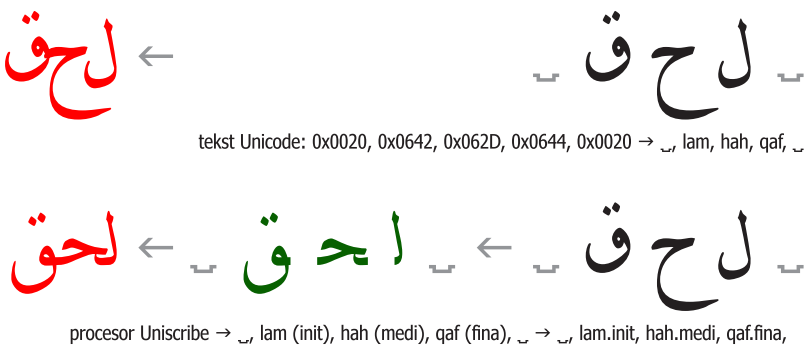


Rys. 20. Dostęp do funkcji zecerskich OpenType w programie Apple Pages pod Mac OS X 10.4 (Tiger).

W dużym uproszczeniu, do tego celu wymyślone zostały funkcje zecerskie OpenType. Wcześniej, przed opracowaniem tego formatu, nie istniał żaden wspólny system kodowania czy składania w tych językach. Istniały specjalizowane, zupełnie niekompatybilne ze sobą rozwiązania. W OpenType zaimplementować można natomiast, w każdym foncie, mechanizmy dające dostęp to tych wszystkich znaków alternatywnych. Robi się to w foncie, gdyż każdy font może mieć inny zestaw znaków i inny repertuar takich funkcji. Oprócz podmiany indywidualnych znaków w foncie OpenType implementuje się też pozycjonowanie akcentów w locie. To znów sprawa bardzo istotna np. dla języka arabskiego. Pismo arabskie to kombinacja „szlaczków” oznaczających spółgłoski i „ptaszków” oznaczających samogłoski. W potocznej

arabszczyźnie zapisuje się tylko spółgłoski, ale w tekstach eleganckich czy oficjalnych, a także oczywiście w Koranie, zapisuje się też samogłoski. W zależności od danej kombinacji spółgłosek w słowie, dokładna pozycja fruwających nad nimi samogłosek może się zmieniać. Można to sobie wyobrazić jako taki kontekstowy kerning w obu wymiarach (*x* i *y*). Informacja o miejscu zawieszenia samogłoski musi się oczywiście różnić w każdym kroju pisma – wszystko zależy od stylu i rysunku danego kroju, od tego czy jest to krój tekstowy i ozdobny, od repertuaru znaków i obsługiwanych języków (np. tylko arabski, czy również perski).

Ponieważ poprawnie pod względem ortograficznym redakcja tekstów w złożonych systemach pisma takich jak arabski wymaga istnienia mechanizmów takich jak kontekstowa zamiana glifów, wstawianie ligatur czy dynamiczne pozycjonowanie znaków względem siebie w dwuwymiarowym układzie współrzędnych, zaimplementowano je w OpenType. Podczas projektowania formatu okazało się, że „tanim kosztem" można jednocześnie dodać funkcje typograficzne pozwalające na uzyskanie eleganckiej typografii również w językach europejskich. Niektóre funkcje typograficzne, takie jak ligatury ozdobne (rys. 22) czy kapitaliki dostępne są w profesjonalnych programach edytorskich takich jak InDesign, gdzie skład tekstu odbywa się pod kontrolą profesjonalnego typografa. Takie funkcje zecerskie można by nazwać „efektowymi".



Rys. 21. Funkcje zecerskie OpenType stosowane są automatycznie dla uzyskania poprawnego ortograficznie składu w różnych językach świata, np. w języku arabskim.

Istnieje jednak cała masa funkcji zecerskich, które mogą, a nawet powinny znaleźć się w każdym edytorze tekstu. Choćby kontekstowy dobór znaków pisarskich, taki jak pokazano na rysunku 23. Jeśli odwrócony znak zapytania używany w języku hiszpańskim pojawi się w składzie wyłącznie wersalikowym, wówczas powinien on zostać podsunięty do góry o dystans zwykle odpowiadający długości wydłużeń dolnych danego kroju pisma. Podobnie o kilka procent powinny zostać podniesione nawiasy. Gdy użytkownik wpisze jakąś cyfrę we frakcji górnej, wówczas – gdy dany font zawiera takie, osobno zaprojektowane cyfry – powinny one zostać zastosowane automatycznie zamiast mechanicznie pomniejszonych cyfr podstawowych (które zwykle po pomniejszeniu okazują się za „lekkie"). Tego typu funkcje zecerskie robią właśnie to, na co wskazuje ich nazwa – zastępują (choćby po części) zawodowego zecera.

Podsumowanie

Poprawna typografia jest tak samo ważna jak poprawna ortografia, choć wielu tego nie zauważa. Poprawna typografia przyspiesza i ułatwia czytanie i rozumienie tekstu. Stawiam tezę, że im lepsza typografia, tym mniej wtórnych analfabetów (oczywiście nie mówimy to o odchyleniach kilkudziesięciu- a jedynie kilkuprocentowych, ale to już coś).

Każdy tekst, jaki czyta przeciętny użytkownik, powinien być złożony i przedstawiony w sposób jak najlepszy. Nie ma znacze-

nia, czy to jest news na stronie internetowej, szyld budki z hamburgerami, menu w restauracji, artykuł w gazecie codziennej czy album poezji. Oczywiście stylistyka formy graficznej powinna oddawać styl przekazu, ale w żadnym wypadku nie powinno dochodzić do kompromisów jakościowych. Nawet jeżeli kupujemy koszulę za 10 zł, a nie za 250, to nie chcemy, by po dwóch tygodniach noszenia urwał nam się guzik. Nie chcemy, by klamka, nawet w toalecie publicznej, urywała się, gdy za nią chwytamy. Podobnie pewien poziom jakości powinien być zachowany w przypadku tekstu.

Tylko wówczas przeciętny klient będzie w stanie docenić wysoką jakość pracy typografa czy operatora DTP. Porównując rynek usług DTP w Niemczech i w Polsce zauważam, że przeciętny klient niemiecki szybciej zauważa, że tekst jest złożony „brzydko". Dlaczego tak się dzieje? To nie kwestia jakichś wrodzonych predyspozycji. Dzieje się tak dlatego, że w swoim życiu przeciętny Niemiec (z Zachodu) częściej niż przeciętny Polak styka się – na co dzień! – z dobrą typografią. Gazety codzienne, beletrystyka, czasopisma, napisy w telewizji, szyldy na stacjach kolejowych, rozkłady jazdy, tablice informacyjne itd. – wszystko to od lat, uśredniając, jest w Niemczech publikowane na wyższym poziomie jakości. Oko ludzkie trenuje się na tym – podświadomie. Nawet jeśli nie potencjalny klient w sposób świadomy nie jest w stanie opisać, co mu się nie podoba, jednak wychwyci typograficzną tandetę. W Polsce ludzie mają w tej chwili znacznie mniej punktów odniesienia. Niewielki procent ludzi miało styczność z naprawdę pięknie wydanymi książkami.

JAN KOTT Jonasz Kofta
Thorgal fjord szufla firet motto

Rys. 22. Z pomocą funkcji zecerskich OpenType ligatury zawarte w foncie są automatycznie podstawiane, gdy użytkownik wpisze odpowiedni tekst.

Dlatego właśnie typografia powinna wchodzić pod strzechy. Typografia to nie jest jakieś fiu-bździu dla zamożnych. Jest to jedna – przyznam, malutka, ale jednak – cegiełka w budowaniu sprawności umysłowej człowieka. Jak już pisałem, stawiam tezę (trudną niestety do udowodnienia), że dobrze złożone książki i gazety powodują, że wiedzę przyswaja się szybciej i łatwiej. Odwracając tę tezę – jeżeli ktoś wychowa się na słowie drukowanym podanym w sposób niechlujny, ta niechlujność w pewnym stopniu odbije się na nim. To trochę jak pylica płuc u górników – nie nabawisz się jej po jednym dniu wizyty w kopalni, ale jeśli przepracujesz tam 20 lat...

W dzisiejszych czasach Internet i komputery stanowią bardzo ważne źródło, z którego szczególnie dzieci i młodzież czerpią wiedzę. Typograficzna jakość prezentacji tekstu w mediach elektronicznych na razie pozostaje daleko za tym, co bardzo dobry typograf może stworzyć na papierze. Ale postęp techniki pozwala na zmiany.

¿DÓNDE USTED DESEA IR HOY?
(SOMEPLACE WITH OPENTYPE)
¿¿H?? ((H))

Rys. 23. Funkcje zecerskie OpenType umożliwiają optyczną korektę znaków interpunkcyjnych w zależności od tego, czy tekst składany jest wersalikami, czy minuskułą.

Podam przykład: Microsoft Word posiada sprytny mechanizm automatycznego rozpoznawania języka, w którym napisany jest dany tekst. Jeżeli piszemy tekst po polsku, naciśniemy klawisz cudzysłowu, wpiszemy jakieś słowo i naciśniemy klawisz cudzysłowu ponownie, Word automatycznie poprawi znaki cala na prawdziwe polskie cudzysłowy. Podobnie stanie się z tekstem angielskim, czeskim, niemieckim czy francuskim. Gdyby funkcja ta wymagała jakichś ręcznych ustawień, wówczas 99% dokumentów tworzonych w Wordzie straszyłaby obrzydliwymi zna-

kami cala. Po 10-15 latach stałoby się to normą i piękna, a przede wszystkim pożyteczna tradycja poszłaby w niepamięć. Podobnie, Word sprytnie zamienia dywizy na półpauzy. W przyszłych wersjach będzie też wstawiał pasujące pod względem typograficznym znaki we frakcji górnej i dolnej, korygował położenie znaków pisarskich (np. nawiasów) w zależności od tego, czy tekst pisany jest wersalikami czy normalnie, automatycznie korzystał z kerningu i tak dalej.

Ponieważ Microsoft (a także Apple) obsługę funkcji zecerskich wbudowują na poziomie systemu operacyjnego, można przyjąć, że zabiegi te wykonywał będzie nie tylko Word, ale także PowerPoint, Keynote, Internet Explorer, Safari, Outlook czy Mail. Gdy użytkownik przyzwyczai się do tego, że komputer bez jego wiedzy i ingerencji serwuje mu porządnie złożony tekst, wówczas łatwiej mu będzie dojrzeć typograficzne babole. No i – w przypadku usług DTP – zacznie rozróżniać między tym, który produkuje typograficzne buble a tym, który zna się na rzeczy.

Adam Twardoch jest konsultantem typograficznym specjalizującym się w problematyce typografii wielojęzycznej i fontów komputerowych.

Pracuje m.in. dla Adobe, Bitstream, Fontlab Ltd., Linotype, MyFonts. com, jest też delegatem krajowym Polski i członkiem zarządu w międzynarodowym stowarzyszeniu typograficznym ATypI.

Jego strona internetowa to http://www.twardoch.com/